REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
11-01-2022	Final Report	25-Sep-2017 - 24-Apr-2021

4. TITLE AND SUBTITLE
Final Report: Advancing Human and Machine Question Answering via Human-Machine Collaboration

5a. CONTRACT NUMBER
W911NF-17-1-0412

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
611102

6. AUTHORS

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
Ohio State University
1960 Kenny Road

Columbus, OH 43210 -1016

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
71694-MA.17

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Huan Sun
UU	UU	UU	UU		19b. TELEPHONE NUMBER: 614-688-1078

RPPR Final Report
as of 11-Jan-2022

Agency Code: 21XD

Proposal Number: 71694MA **Agreement Number: W911NF-17-1-0412**
INVESTIGATOR(S):

Name: Huan Sun
Email: sun.397@osu.edu
Phone Number: 6146881078
Principal: Y

Organization: **Ohio State University**
Address: 1960 Kenny Road, Columbus, OH 432101016
Country: USA
DUNS Number: 832127323 EIN: 316025986
Report Date: 24-Jul-2021 Date Received: 11-Jan-2022
Final Report for Period Beginning 25-Sep-2017 and Ending 24-Apr-2021
Title: Advancing Human and Machine Question Answering via Human-Machine Collaboration
Begin Performance Period: 25-Sep-2017 **End Performance Period:** 24-Apr-2021
Report Term: 0-Other
Submitted By: Huan Sun Email: sun.397@osu.edu
Phone: (614) 688-1078
Distribution Statement: 1-Approved for public release; distribution is unlimited.

STEM Degrees: 5 **STEM Participants:** 8

Major Goals: There are two main channels for a human to obtain answers to her questions: one is to resort to machines like search engines, and the other is to resort to humans such as experts in online forums. We aim to develop novel human-machine collaboration mechanisms to facilitate both question answering (QA) channels. While most existing studies focus on either machines or humans, we observe that the capabilities of humans and machines are highly complementary to each other. The respective limitations of current machine and human QA systems can be largely remedied by the other: humans can easily resolve the ambiguities in complex questions that confuse machines, while machines can provide useful insights by analyzing large amounts of data and make human QA systems more automated. Their collaboration has the potential to greatly advance both QA channels.

The overall goal of this project is to explore human-machine collaboration mechanisms that can address the challenges faced by the cutting-edge machine and human QA systems. Towards this goal, we propose innovative techniques that foster the synergies among humans and machines for question answering. Our grand vision is that humans and machines should team up as an integrated complex system for effective and efficient QA. The techniques proposed in this project can potentially be generalized to many other scenarios where human and machine intelligence need to be combined in an algorithmic manner.

Accomplishments: In this project, we aim to explore human-machine collaboration mechanisms that can address the challenges faced by the cutting-edge machine and human QA systems. Towards this end, we have proposed innovative techniques to involve humans in the loop for machine QA systems or developed advanced algorithms to boost the efficiency of human QA systems. Here we select a few publications to highlight some of our achievements. Details can be found in the uploaded pdf.

1. Involve humans in the loop for machine QA systems and natural language interfaces in general.

Natural language interfaces (NLIs) allow users to query data in natural language and command machines without programming. Current commercial products that heavily use NLI technologies include Amazon Alexa, Apple Siri, Google home, etc. There are many challenges for building or scaling up NLIs: (1) high bootstrapping cost, especially for specialized domains, because mainstream deep learning models are data-hungry and annotation cost of data is relatively high (as domain expertise is often required), (2) high fine-tuning cost from continuously analyzing usage and annotating new data, and (3) privacy risks arising from exposing private user conversations to annotators and developers. We have been pushing the state of the arts in addressing all these challenges, e.g., by developing automatic data collection methods to reduce bootstrapping cost (e.g., WWW'18 and EMNLP'20) and exploring an emerging direction of interactive semantic parsing with human users in the loop (e.g., EMNLP'19 and

EMNLP'20). Beyond question answering, we investigated interactive semantic parsing for program synthesis (AAAI'19), where the agent can ask the user clarification questions to resolve ambiguities via a multi-turn dialogue, on an important type of programs called "If-Then recipes." We develop a hierarchical reinforcement learning (HRL) based agent that significantly improves the parsing performance with minimal questions to the user.

2. Question/answer retrieval to boost the efficiency of human QA forums.

Online forums such as Stack Overflow have contributed a huge number of code snippets, understanding and reuse of which can greatly speed up software development and help users search the forum and retrieve relevant questions and answers. Towards this goal, a lot of research work have been developed recently, such as retrieving or generating code snippets based on a natural language query, and annotating code snippets using natural language. At the core of these work are machine learning models that map between natural language and programming language, which are typically data-hungry and require large-scale and high-quality <natural language question, code solution> pairs (i.e., question-code or QC pairs). In this project, we investigated systematically mining large-scale high-quality question-code pairs, and presented StaQC (WWW'18), the largest dataset at of ~148K Python and ~120K SQL question-code pairs, systematically mined by our framework. Given such a dataset, code retrieval (CR) and code annotation (CA) are two important tasks that have been widely studied in the past few years, where the former aims to retrieve relevant code snippets based on a natural language (NL) query while the latter is to generate natural language descriptions to describe what a given code snippet does. In WWW'19, we explored a novel perspective - code annotation *for* code retrieval (CoaCor), which is to generate an NL annotation for a code snippet so that the generated annotation can be used for code retrieval (i.e., can represent the semantic meaning of a code snippet and distinguish it from others w.r.t. a given NL query in the code retrieval task). We developed an effective RL-based framework with a novel rewarding mechanism, in which a code retrieval model is directly used to formulate rewards and guide the annotation generation. In EMNLP'20, we further studied code retrieval, where we directly match a code snippet with a given question, but during training, we will utilize question-description relevance to improve the learning process. We were the first to introduce adversarial training to code retrieval, given its success on transfer learning from one domain to another and learning with scarce supervised data.

3. Representation learning with pre-trained language models for information extraction (from semi-structured data) and QA.

Relational tables on the Web store a vast amount of knowledge. Owing to the wealth of such tables, there has been tremendous progress on a variety of tasks in the area of table understanding (such as information extraction from tables). However, existing work generally relies on heavily-engineered task-specific features and model architectures. In this project, we presented TURL (VLDB'21), a novel framework that introduces the pre-training/fine-tuning paradigm to relational Web tables. During pre-training, our framework learns deep contextualized representations on relational tables in an unsupervised manner. Its universal model design with pre-trained representations can be applied to a wide range of tasks with minimal task-specific fine-tuning. Specifically, we proposed a structure-aware Transformer encoder to model the row-column structure of relational tables, and presented a new Masked Entity Recovery (MER) objective for pre-training to capture the semantics and knowledge in large-scale unlabeled data. We systematically evaluated TURL with a benchmark consisting of 6 different tasks for table understanding (e.g., relation extraction, cell filling). We showed that TURL generalizes well to all tasks and substantially outperforms existing methods in almost all instances.

We further studied how to augment language models with the ability to reason over long-range relations and multiple, possibly hybrid contexts and presented a pre-training method dubbed ReasonBert in EMNLP'21. Unlike existing pre-training methods that only harvest learning signals from local contexts of naturally occurring texts, we proposed a generalized notion of distant supervision to automatically connect multiple pieces of text and tables to create pre-training examples that require long-range reasoning. Different types of reasoning were simulated, including intersecting multiple pieces of evidence, bridging from one piece of evidence to another, and detecting unanswerable cases. We conducted a comprehensive evaluation on a variety of extractive QA datasets ranging from single-hop to multi-hop and from text-only to table-only to hybrid that require various reasoning capabilities and show that ReasonBert achieves remarkable improvement over an array of strong baselines. Few-shot experiments further demonstrated that our pre-training method substantially improves sample efficiency.

RPPR Final Report
as of 11-Jan-2022

Training Opportunities: (1) Training doctoral students. Ziyu Yao who was mainly supported by this ARO grant during her Ph.D. study graduated in Summer 2021 and joined George Mason University as an assistant professor. Her thesis is “On Advancing Natural Language Interfaces: Data Collection, Model Development, and User Interaction.”

Jie Zhao who was also mainly supported by this ARO grant during his Ph.D. study graduated in Fall 2020 and joined Amazon as an applied scientist. His thesis is “Towards Building Trustworthy Automatic Question Answering Systems.”

Bortik Bandyopadhyay (partly supported by the PI using her NSF
grant) graduated in Summer 2020 and joined Apple to work on machine learning and AI. While Bortik was not directly supported by this ARO grant, his work on "Automatic Table Completion using Knowledge Base" included in his dissertation is directly related to machine intelligent question answering.

(2) Training undergraduate and high school students. The PI has mentored many undergraduate students since the start of this project. In particular, undergraduate students Frederick Zhang and Heming Sun worked on projects related to this ARO project. They had published a paper in EMNLP'21 on Covid-related QA, where questions were crawled from various webpages and community QA forums. Frederick also co-led another project on question generation for domain adaptation of clinical QA systems, which won the Best Paper Award at BIBM'21. These pieces of work help boost the impact of the ARO project on clinical applications. The two students joined University of Michigan and University of Southern California for graduate studies in Fall 2021.

In addition, the PI also closely worked with two high school interns during 06-08/2018: Sounak Dey from Thomas Worthington High School and Jacob Mayhew from St. Charles Preparatory School.

(3) Training master students. The PI has closely worked with a few master students including Kaushik Mani and Jayavardhan Reddy Peddamail, who published papers related to the ARO project and joined Oracle and Google, respectively, after graduation.

RPPR Final Report
as of 11-Jan-2022

Results Dissemination: A. During the reporting period, we have produced the following publications in top-tier conferences, on which one or more authors (including the PI) are partly funded by the grant:

1. Xiang Deng, Yu Su, Alyssa Lees, You Wu, Cong Yu, Huan Sun, “ReasonBERT: Pre-trained to Reason with Distant Supervision,” The 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021, long paper).

2. Xiang Deng, Huan Sun, Alyssa Lees, You Wu, Cong Yu, “TURL: Table Understanding through Representation Learning,” The 47th International Conference on Very Large Data Bases (VLDB'21).

3. Zhen Wang, Jennifer Lee, Simon Lin, Huan Sun, “Rationalizing Medical Relation Prediction from Corpus-level Statistics,” Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL'20, long).

4. Xiang Yue, Bernal Jimenez, Huan Sun, “Clinical Reading Comprehension: A Thorough Analysis of the emrQA Dataset,” Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL'20, long).

5. Bernhard Kratzwald, Huan Sun, Stefan Feuerriegel, "Learning a Cost-Effective Annotation Policy for Question Answering," 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP'20).

6. Jie Zhao, Huan Sun, "Adversarial Training for Code Retrieval with Question-Description Relevance Regularization," 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP'20 Findings).

7. Ziyu Yao, Yiqi Tang, Wen-tau Yih, Huan Sun and Yu Su, "An Imitation Game for Learning Semantic Parsers from User Interaction," 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP'20).

8. Ziyu Yao, Yu Su, Huan Sun, Wen-tau Yih, “Model-based Interactive Semantic Parsing: A Unified Formulation and A Text-to-SQL Case Study,” 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP'19).

9. Xiang Deng, Huan Sun, “Leveraging 2-hop Distant Supervision from Table Entity Pairs for Relation Extraction,” 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP'19).

10. Boyuan Pan, Hao Li, Ziyu Yao, Deng Cai, Huan Sun, “Reinforced Dynamic Reasoning for Conversational Question Generation,” Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL'19).

11. Jie Zhao, Ziyu Guan, Huan Sun, “Riker: Mining Rich Keyword Representations for Interpretable Product Question Answering,” The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2019 (SIGKDD'19, research track, acceptance rate: ~14.2%, poster). (SIGKDD'19: ~110 oral + ~60 poster presentations selected from ~1200 submissions)

12. Zhen Wang, Xiang Yue, Soheil Moosavinasab, Yungui Huang, Simon Lin, Huan Sun, “SurfCon: Synonym Discovery on Privacy-Aware Clinical Data,” The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2019 (SIGKDD'19, research track, acceptance rate: ~14.2%, oral).

13. Ziyu Yao, Jayavardhan Reddy Peddamail, Huan Sun, “CoaCor: Code Annotation for Code Retrieval with Reinforcement Learning,” The Web Conference (former WWW Conference) 2019 (WWW'19, acceptance rate: 18%, Oral + Poster).

14. Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, Huan Sun, “Interactive Semantic Parsing for If-Then Recipes via Hierarchical Reinforcement Learning,” The AAAI Conference on Artificial Intelligence 2019 (AAAI’19, acceptance rate: 16.2%).

15. Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, Huan Sun, “StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow,” The Web Conference (former WWW Conference) 2018 (WWW'18, acceptance rate: 14.8%).

B. During the reporting period, the following efforts were made to disseminate our work:

RPPR Final Report
as of 11-Jan-2022

2020-2021:
1. PI Sun and her student Xiang Deng presented their work at NAACL, Online, June 2021.
2. PI Sun and her student Xiang Deng presented their work at VLDB, Online, August 2021.
3. PI Sun and her student Ziyu Yao presented their work at ICLR, Online, April 2021.
4. PI Sun gave an invited talk at Nesh (Startup), Texas AI community, Online, July 2021.
5. PI Sun co-presented a tutorial at SIGKDD, Online, August 2021.
6. PI Sun gave an invited talk at Allen Institute for AI (AI2), Online, July 2021.
7. PI Sun gave an invited talk at Facebook AI Research, Online, November 2020.
8. PI Sun gave an invited talk at UIUC, online, October 2020.

2019-2020:
1. PI Sun and her student Zhen Wang presented their work at ACL, Online, July 2020.
2. PI Sun and her student Tommy Yue presented their work at ACL, Online, July 2020.
3. PI Sun's student Ziyu Yao gave an invited talk at CMU, Online, May 2020.
4. PI Sun's student Xiang Deng gave an invited talk at MSR, Online, Summer 2020.
5. PI Sun gave an invited talk at Rulai (Startup), Online, June 2020.
6. PI Sun gave an invited talk at Google Research, New York, Feb 2020.
7. PI Sun gave an invited talk to the Facebook Conversational AI group, Menlo Park, Oct 2019.

2018-2019:
1. PI Sun and her student Ziyu Yao presented their work at AAAI, Hawaii, USA, in Jan 2019.
2. PI Sun gave an invited talk at Army Science Planning and Strategy Meeting, Austin, Texas, in Jan 2019.
3. PI Sun gave an invited talk at OSU-Tsukuba Joint Linguistics Workshop, Columbus, Ohio, in February 2019.
4. PI Sun and her student Ziyu Yao presented their work at WWW, SF, USA, in May 2019.
5. PI Sun gave an invited talk at IEEE National Aerospace & Electronics Conference (NAECON), Dayton, Ohio, in July 2019.
6. PI Sun's student Ziyu Yao presented their work at ACL, Florence, Italy, in July 2019.
7. PI Sun's student Ziyu Yao gave an invited talk at ETH, Zurich, in July 2019.
8. PI Sun and her students Jie Zhao, Xiang Yue, Zhen Wang presented their KDD work at SIGKDD, Anchorage, USA, in August 2019.
9. PI Sun gave an invited talk at Fujitsu Laboratories of America in Sunnyvale, CA, in August 2019.

2017-2018:
1. PI Sun’s student Ziyu Yao gave 2 departmental talks respectively on the work published in WWW'18 and in AAAI'19.
2. PI Sun’s student Ziyu Yao gave 1 conference presentation at WWW’18, Lyon, France, in April 2018.
3. During Jan 2018 ~ July 2018, PI Sun gave 7 invited talks respectively at University of Illinois, Chicago, Penn State University, Rice University, UMass Amherst, Tsinghua University, Fudan University, and City Brain Project at Hangzhou, China.
4. PI Sun's student Jayavardhan Reddy Peddamail gave 1 spotlight presentation at SIGKDD’18, London, UK, 08/20/2018.

RPPR Final Report
as of 11-Jan-2022

Honors and Awards: 1. PI Sun and her students received the Best Paper Award at the 2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM, long paper).

2. Ph.D. student Ziyu Yao (who was mainly supported by this ARO grant) won the prestigious Presidential Fellowship in Fall 2020. She was selected into EECS Rising Stars 2020. Ziyu also won the Graduate Research Award by CSE Department at OSU in 2021. She joined George Mason University as a tenure-track assistant professor in Fall 2021.

Ph.D. student Zhen Wang (who was partly supported by this ARO grant) was selected to attend the 2021 Rising Stars in Data Science Workshop hosted by UChicago.

3. Undergraduate student Xinliang Frederick Zhang received the Honorable Mention of Outstanding Undergraduate Researcher Award 2021 by CRA (Computing Research Association). He also won the Undergraduate Research Award by CSE Department at OSU in 2021. He joined University of Michigan for Ph.D. studies in Fall 2021.

4. PI Sun received NSF CAREER Award, Google Faculty Research Award, Lumley Research Award, all in 2020.

5. PI Sun's team was selected to participate in the first Amazon Alexa Taskbot Challenge (10 out of 125 applicant teams), because of the series work published under the support of this ARO grant.

Protocol Activity Status:

Technology Transfer: Nothing to Report

PARTICIPANTS:

Participant Type: Graduate Student (research assistant)
Participant: Ziyu Yao
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Jie Zhao
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Zhen Wang
Person Months Worked: 5.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Xiang Yue
Person Months Worked: 5.00 **Funding Support:**
Project Contribution:
National Academy Member: N

RPPR Final Report
as of 11-Jan-2022

Participant Type: Graduate Student (research assistant)
Participant: Xiang Deng
Person Months Worked: 9.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: PD/PI
Participant: Huan Sun
Person Months Worked: 2.00 **Funding Support:**
Project Contribution:
National Academy Member: N

CONFERENCE PAPERS:

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: The Web Conference (former WWW Conference) 2018
Date Received: 29-Aug-2018 Conference Date: 23-Apr-2018 Date Published:
Conference Location: Lyon, France
Paper Title: StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow
Authors: Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, Huan Sun
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: SIGKDD 2018 Deep Learning Day
Date Received: 29-Aug-2018 Conference Date: 20-Aug-2018 Date Published:
Conference Location: London, UK
Paper Title: A Comprehensive Study of StaQC for Deep Code Summarization
Authors: Jayavardhan Reddy Peddamail, Ziyu Yao, Zhen Wang, Huan Sun,
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: The 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP'21)
Date Received: 30-Dec-2021 Conference Date: 07-Nov-2021 Date Published: 11-Nov-2021
Conference Location: Online and Punta Cana, Dominican Republic
Paper Title: ReasonBERT: Pre-trained to Reason with Distant Supervision
Authors: Xiang Deng, Yu Su, Alyssa Lees, You Wu, Cong Yu, Huan Sun
Acknowledged Federal Support: **Y**

RPPR Final Report
as of 11-Jan-2022

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: The AAAI Conference on Artificial Intelligence 2019 (AAAI'19)
Date Received: 05-Jan-2022 Conference Date: 27-Jan-2019 Date Published: 01-Feb-2019
Conference Location: Hawaii
Paper Title: Interactive Semantic Parsing for If-Then Recipes via Hierarchical Reinforcement Learning
Authors: Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, Huan Sun
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: The Web Conference (former WWW Conference) 2019
Date Received: 05-Jan-2022 Conference Date: 13-May-2019 Date Published: 17-May-2019
Conference Location: San Francisco
Paper Title: CoaCor: Code Annotation for Code Retrieval with Reinforcement Learning
Authors: Ziyu Yao, Jayavardhan Reddy Peddamail, Huan Sun
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2019 (SIGKDD'19)
Date Received: 05-Jan-2022 Conference Date: 04-Aug-2019 Date Published: 08-Aug-2019
Conference Location: Anchorage
Paper Title: SurfCon: Synonym Discovery on Privacy-Aware Clinical Data
Authors: Zhen Wang, Xiang Yue, Soheil Moosavinasab, Yungui Huang, Simon Lin, Huan Sun
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2019 (SIGKDD'19)
Date Received: 05-Jan-2022 Conference Date: 04-Aug-2019 Date Published: 08-Aug-2019
Conference Location: Anchorage
Paper Title: Riker: Mining Rich Keyword Representations for Interpretable Product Question Answering
Authors: Jie Zhao, Ziyu Guan, Huan Sun
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL'19)
Date Received: 05-Jan-2022 Conference Date: 28-Jul-2019 Date Published: 02-Aug-2019
Conference Location: Florence, Italy
Paper Title: Reinforced Dynamic Reasoning for Conversational Question Generation
Authors: Boyuan Pan, Hao Li, Ziyu Yao, Deng Cai, Huan Sun
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP'19)
Date Received: 05-Jan-2022 Conference Date: 03-Nov-2019 Date Published: 07-Nov-2019
Conference Location: Hong Kong, China
Paper Title: Leveraging 2-hop Distant Supervision from Table Entity Pairs for Relation Extraction
Authors: Xiang Deng, Huan Sun
Acknowledged Federal Support: **Y**

RPPR Final Report
as of 11-Jan-2022

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP'19)
Date Received: 05-Jan-2022 Conference Date: 03-Nov-2019 Date Published: 07-Nov-2019
Conference Location: Hong Kong, China
Paper Title: Model-based Interactive Semantic Parsing: A Unified Formulation and A Text-to-SQL Case Study
Authors: Ziyu Yao, Yu Su, Huan Sun, Wen-tau Yih
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP'20)
Date Received: 05-Jan-2022 Conference Date: 16-Nov-2020 Date Published: 20-Nov-2020
Conference Location: Virtual
Paper Title: An Imitation Game for Learning Semantic Parsers from User Interaction
Authors: Ziyu Yao, Yiqi Tang, Wen-tau Yih, Huan Sun and Yu Su
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: Findings of 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP'20 Findings)
Date Received: 05-Jan-2022 Conference Date: 16-Nov-2020 Date Published: 20-Nov-2020
Conference Location: Virtual
Paper Title: Adversarial Training for Code Retrieval with Question-Description Relevance Regularization
Authors: Jie Zhao, Huan Sun
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP'20)
Date Received: 05-Jan-2022 Conference Date: 16-Nov-2020 Date Published: 20-Nov-2020
Conference Location: Virtual
Paper Title: Learning a Cost-Effective Annotation Policy for Question Answering
Authors: Bernhard Kratzwald, Huan Sun, Stefan Feuerriegel
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL'20)
Date Received: 05-Jan-2022 Conference Date: 05-Jul-2020 Date Published: 10-Jul-2020
Conference Location: Virtual
Paper Title: Clinical Reading Comprehension: A Thorough Analysis of the emrQA Dataset
Authors: Xiang Yue, Bernal Jimenez, Huan Sun
Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL'20)
Date Received: 05-Jan-2022 Conference Date: 05-Jul-2020 Date Published: 10-Jul-2020
Conference Location: Virtual
Paper Title: Rationalizing Medical Relation Prediction from Corpus-level Statistics
Authors: Zhen Wang, Jennifer Lee, Simon Lin, Huan Sun
Acknowledged Federal Support: **Y**

RPPR Final Report
as of 11-Jan-2022

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: The 47th International Conference on Very Large Data Bases (VLDB'21)
Date Received: 05-Jan-2022 Conference Date: 16-Aug-2021 Date Published: 20-Aug-2021
Conference Location: Copenhagen, Denmark
Paper Title: TURL: Table Understanding through Representation Learning
Authors: Xiang Deng, Huan Sun, Alyssa Lees, You Wu, Cong Yu
Acknowledged Federal Support: **Y**

Partners

,

1. Army Research Lab (ARL), Adelphi, MarylandDr. Brian Sadler from ARL provided comments on the interactive se

I certify that the information in the report is complete and accurate:
Signature: Huan Sun
Signature Date: 1/11/22 9:38AM

StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow

Ziyu Yao[†], Daniel S. Weld[#], Wei-Peng Chen[††], Huan Sun[†]

[†]The Ohio State University, [#]University of Washington, [††]Fujitsu Labs of America
{yao.470,sun.397}@osu.edu,weld@cs.washington.edu,wei-peng.chen@us.fujitsu.com

ABSTRACT

Stack Overflow (SO) has been a great source of natural language questions and their code solutions (i.e., question-code pairs), which are critical for many tasks including code retrieval and annotation. In most existing research, question-code pairs were collected *heuristically* and tend to have low quality. In this paper, we investigate a new problem of *systematically mining* question-code pairs from Stack Overflow (in contrast to *heuristically collecting* them). It is formulated as predicting whether or not a code snippet is a standalone solution to a question. We propose a novel Bi-View Hierarchical Neural Network which can capture both the programming content and the textual context of a code snippet (i.e., two views) to make a prediction. On two manually annotated datasets in Python and SQL domain, our framework substantially outperforms heuristic methods with at least **15%** higher F_1 and accuracy. Furthermore, we present StaQC (Stack Overflow Question-Code pairs), the largest dataset to date of ~**148K** Python and ~**120K** SQL question-code pairs, automatically mined from SO using our framework. Under various case studies, we demonstrate that StaQC can greatly help develop data-hungry models for associating natural language with programming language[1].

CCS CONCEPTS

• **Information systems** → **Web searching and information discovery**; **Question answering**; **Information extraction**; • **Software and its engineering** → *Automatic programming*;

KEYWORDS

Natural Language Question Answering; Question-Code Pairs; Deep Neural Networks; Stack Overflow

ACM Reference Format:
Ziyu Yao[†], Daniel S. Weld[#], Wei-Peng Chen[††], Huan Sun[†] . 2018. StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3178876.3186081

[1]Available at https://github.com/LittleYUYU/StackOverflow-Question-Code-Dataset.

WWW 2018, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186081

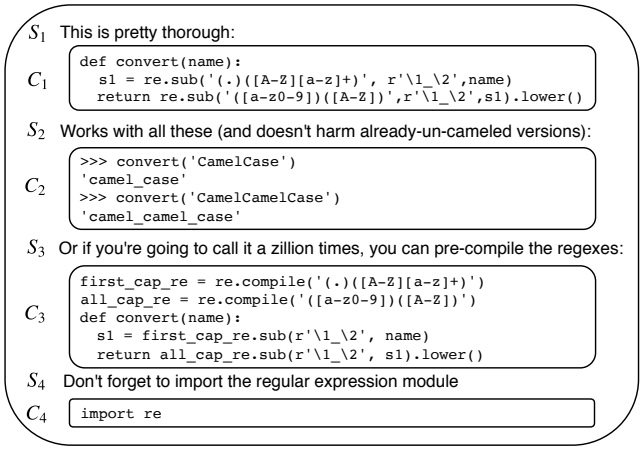


Figure 1: The accepted answer post to question *"Elegant Python function to convert CamelCase to snake_case?"* in SO. S_i ($i = 1, 2, 3, 4$) and C_j ($j = 1, 2, 3, 4$) denote sentence blocks and code blocks respectively, which can be trivially separated based on the HTML format.

1 INTRODUCTION

Online forums such as Stack Overflow (SO) [40] have contributed a huge number of code snippets, understanding and reuse of which can greatly speed up software development. Towards this goal, a lot of research work have been developed recently, such as retrieving or generating code snippets based on a natural language query, and annotating code snippets using natural language [2, 21, 28, 44, 52, 61, 64]. At the core of these work are machine learning models that map between natural language and programming language, which are typically data-hungry [19, 24, 46] and require large-scale and high-quality <natural language question, code solution> pairs (i.e., question-code or QC pairs).

In our work, *we define a code snippet as a code solution when the questioner can solve the problem solely based on it* (also named as "standalone" solution). Take Figure 1 as an example, which shows the accepted answer post[2] to question "*Elegant Python function to convert CamelCase to snake_case*". Among the four code snippets $\{C_1, C_2, C_3, C_4\}$, only C_1 and C_3 are standalone code solutions to

[2]In SO, an accepted answer post is marked with a green check by the questioner, if he/she thinks it solves the problem. Following previous work [21, 59], although there can be multiple answer posts to a question, *we only consider the accepted one because of its verified quality*, and use "accepted answer post" and "answer post" interchangeably.

the question while the rest are not, because C_2 only gives an input-output demo of the "convert" function without its definition and C_4 is a reminder of an additional detail. Given an answer post with multiple code snippets (i.e., a multi-code answer post) like Figure 1, previous work usually collected question-code pairs in heuristic ways: Simply pair the question title with the first code snippet, or with each code snippet, or with the concatenation of all code snippets in the post [2, 64]. Iyer et al. [21] merely employed accepted answer posts that contain exactly one code snippet, and discarded all others with multiple code snippets. Such heuristic question-code collection methods suffer from at least one of the following weaknesses: (1) Low precision: Questions do not match with their paired code snippets, when the latter serve as background, explanation, or input-output demo rather than as a solution (e.g., C_2 in Figure 1); (2) Low recall: If one only selects the first code snippet to pair with a question, other code solutions in an answer post (e.g., C_3) will be unemployed.

In fact, multi-code answer posts are very common in SO, which makes the low-precision and low-recall issues even more prominent. In the Stack Exchange Data dump[51], among all accepted answer posts for Python and SQL "how-to-do-it" questions (to be introduced in Section 2), 44.66% and 34.35% contain more than one code snippets respectively. Note that an accepted answer post was verified only *as an entirety* by the questioner, and labels on whether each individual code snippet serves as a standalone solution or not are not readily available. Moreover, it is not feasible to obtain such labels by simply running each code snippet in a programming environment for two reasons: (1) A runnable code snippet is not necessarily a code solution (e.g., C_4 in Figure 1); (2) It was reported that around 74% of Python and 88% of SQL code snippets in SO are not directly parsable or runnable [21, 59]. Nevertheless, many of them usually contain critical information to answer a question. Therefore, they can still be used in semantic analysis for downstream tasks [2, 21, 59, 64] once paired with natural language questions.

To systematically mine question-code pairs with high precision and recall, we propose a novel task: *Given a question*[3] *in SO and its accepted answer post with multiple code snippets, how to predict whether each code snippet is a standalone solution or not?* In this paper, we focus on "how-to-do-it"-type of questions which ask how to implement a certain task like in Figure 1, since answers to such questions are most likely to be standalone code solutions. The definition and classification of different types of questions will be discussed in Section 2. We identify two challenges in our task: (1) As shown in Figure 1, code snippets in an answer post can play many non-solution roles such as serving as an input-output demo or reminder (e.g., C_2 and C_4), which calls for a statistical learning model to make accurate predictions. (2) Both the textual context and the programming content of a code snippet can be predictive, but an effective model to jointly utilize them needs careful design. Intuitively, a text block with patterns like "you can do ..." and "this is one thorough solution ..." is more likely to be followed by a code solution. For example, given S_1 and S_3 in Figure 1, a code solution is likely to be introduced after them. On the other hand, by inspecting the code content, C_2 is probably not a code solution to the question, since it contains special Python console patterns like ">>> ... >>>" and no particular definition of "convert".

To tackle these challenges, we explore a series of models including traditional classifiers and deep learning models, and propose a novel model, named Bi-View Hierarchical Neural Network (BiV-HNN), to capture both the textual context and the programming content of each code snippet (which make the two views). In BiV-HNN, we design two different modules to learn features from text and code respectively, and combine them into a deep neural network architecture, which finally predicts whether a code snippet is a standalone solution or not. To summarize, our contributions lie in three folds:

First, to the best of our knowledge, we are the first to investigate *systematically mining large-scale high-quality question-code pairs*, which are critical for developing learning-based models aiming to map between natural language and programming language.

Second, we extensively explore various models including traditional classifiers and deep learning models to predict whether a code snippet is a solution or not, and propose a novel Bi-View Hierarchical Neural Network which considers both text- and code-based views. On two manually labeled datasets in Python and SQL domain, BiV-HNN outperforms both the widely adopted heuristic methods and traditional classifiers by a large margin in terms of F_1 and accuracy. Moreover, BiV-HNN does not rely on any prior knowledge and can be easily applied to other programming domains.

Last but not least, we present StaQC, the largest dataset to date of ∼**148K** Python and ∼**120K** SQL question-code pairs, systematically mined by our framework. Using multiple case studies, we show that (1) StaQC is rich in surface variation: A question can be paired with multiple code solutions, and semantically the same code snippets can have different/paraphrased natural language descriptions. (2) Owing to such diversity as well as its large scale, StaQC is a much better data resource than existing ones for constructing models to map between natural language and programming language. In addition, we can continue to grow StaQC in both size and diversity, by regularly applying our framework to the fast-growing SO. Question-code pairs in other programming languages can also be mined similarly and included in StaQC.

2 PRELIMINARIES

In this section, we first clarify our task definition, and then describe how we annotated datasets for model development.

2.1 Task Definition

Given a question and its accepted answer post which contains multiple code snippets in Stack Overflow, we aim at predicting whether each code snippet in the answer post is a *standalone* solution to the question or not. As explained in Section 1, we focus on "accepted" answer posts and "standalone" solutions.

Users can ask different types of questions in SO such as "how to implement X" and "what/why is Y". Following previous work [12, 13, 32], we divide questions into five types: "How-to-do-it", "Debug/corrective", "Conceptual", "Seeking something, e.g., advice, tutorial", and their combinations. In particular, a question is of type "how-to-do-it" when the questioner provides a scenario and asks how to implement it like in Figure 1.

[3]Following previous work [2, 6, 21], we only use the title of a question post in this work, and leave incorporating the question post content for future work.

For collecting question-code pairs, we target at "how-to-do-it" questions, because answers to other types of questions are not very likely to be standalone code solutions (e.g., answers to "Conceptual" questions are usually text descriptions). Next, we describe how to distinguish "how-to-do-it" questions from others.

2.2 "How-to-do-it" Question Collection

2.2.1 Question Type Classification.

At the high level, we combined the other four question types apart from "how-to-do-it" into one category named "non-how-to" and built a binary question type classifier.

We first collected Python and SQL questions from SO based on their tags, which are available for all question posts. Specifically, we considered questions whose tags contain the keyword "python" to be in Python domain and questions tagged by "sql", "database" or "oracle" to be in SQL domain. For each domain, we randomly sampled and labeled 250 questions for training (150), validating (20) and testing (80) the classifier[4]. Among the 250 questions, around 45% in Python and 57% in SQL are "how-to-do-it" questions. We built one Logistic Regression classifier respectively for each domain, based on simple features extracted from question and answer posts as in [13], such as keyword-occurrence features, the number of code blocks in question/answer posts, the maximum length of code blocks, etc. Hyperparameters in classifiers were tuned based on validation sets.

Finally, we obtained a question-type classification accuracy of 0.738 (precision: 0.653, recall: 0.889, and F_1: 0.753) for Python and an accuracy of 0.713 (precision: 0.625, recall: 0.946, and F_1: 0.753) for SQL. The classification of question types may be further improved with more advanced features and algorithms, which is not the focus of this paper.

2.2.2 "How-to-do-it" Question Set Collection.

Using the above classifiers, we classified all Python and SQL questions in SO whose accepted answer post contains code blocks and collected a large set of "how-to-do-it" questions in each domain. Among these "how-to-do-it" questions, around 44.66% (68,839) Python questions and 34.45% (39,752) SQL questions have an accepted answer post with more than one code snippets, from which we will systematically mine question-code pairs.

2.3 Annotating QC Pairs for Model Training

To construct training/validation/testing datasets for our task, we hired four undergraduate students familiar with Python and SQL to annotate answer posts in these two domains. For each code snippet in an answer post, annotators can assign "1" to it if they think they can solve the problem based on the code snippet alone (i.e., it is a standalone code solution), and "0" otherwise. We ensured each code snippet is annotated by two annotators and adopted the label only when both annotators agreed on it. For each programming language, around 85% code snippets were labeled. The average Cohen's kappa agreement [9] is around 0.658 for Python and 0.691 for SQL. The statistics of our manually annotated datasets are summarized in Table 1, which will be used to develop our models.

[4]Despite of the small amount of training data, no overfitting was observed in our experiments partly because the features are very simple.

	Python		SQL	
	# of QC pairs	% of QC pairs with label "1"	# of QC pairs	% of QC pairs with label "1"
Training	2,932	43.89%	2,183	56.12%
Validation	976	43.14%	727	55.98%
Testing	976	47.23%	727	58.32%

Table 1: Statistics of manually annotated datasets.

3 BI-VIEW HIERARCHICAL NN

Without loss of generality, let us assume an answer post of a given question has a sequence of *blocks* $\{S_1, C_1, S_2, ..., S_i, C_i, S_{i+1}, ..., S_{L-1}, C_{L-1}, S_L\}$ with L text blocks (S_i's) and $L-1$ code blocks (C_i's) interleaving with each other. Our task is to automatically assign a binary label to each code snippet C_i, where 1 means a standalone solution while 0 otherwise. In this work, we model each code snippet independently and predict the label of C_i based on its textual context (i.e., S_i, S_{i+1}) and programming content. If either S_i or S_{i+1} is empty, we insert an empty dummy text block to make our model applicable. One can extend our formulation to a more complicated sequence labeling problem where a sequence of code snippets can be modeled simultaneously, which we leave for future work.

3.1 Intuition

We first analyze at the high level how each individual block contributes to elaborating the entire answer fluently. For example, in Figure 1, the first text block S_1 suggests its followed code block C_1 (which implements a function) is "thorough" and thus might be a solution. S_2 subsequently connects C_1 to examples it can work with in C_2. In contrast, S_3 starts with the conjunction word "Or" and possibly will introduce an alternative solution (e.g., C_3). This observation inspires us to first model the meaning of each block separately using a *token-level sequence encoder*, then model the block sequence S_i-C_i-S_{i+1} using a *block-level encoder*, from which we finally obtain the semantic representation of C_i.

Figure 2 shows our model, named Bi-View Hierarchical Neural Network (BiV-HNN). It progressively learns the semantic representation of a code block from token level to block level, based on which we predict it to be a standalone solution or not. On the other hand, BiV-HNN naturally incorporates two views, i.e., textual context and code content, into the model structure. We detail each component as follows.

3.2 Token-level Sequence Encoder

Text block. Given a sentence block S_i with a sequence of words w_{it}, $t \in [1, T_i]$, we first embed the words into vectors through a pretrained word embedding matrix W_e, i.e., $x_{it} = W_e w_{it}$. We then use a bidirectional Gated Recurrent Unit (GRU) based Recurrent Neural Network (RNN) [8] to learn the word representation by summarizing the contextual information from both directions. The GRU tracks the state of sequences by controlling how much information is updated into the new hidden state from previous states. Specifically, given the input word vector x_t in the current step and the hidden state h_{t-1} from the last step, the GRU first computes a *reset gate* r for resetting information from previous steps in order

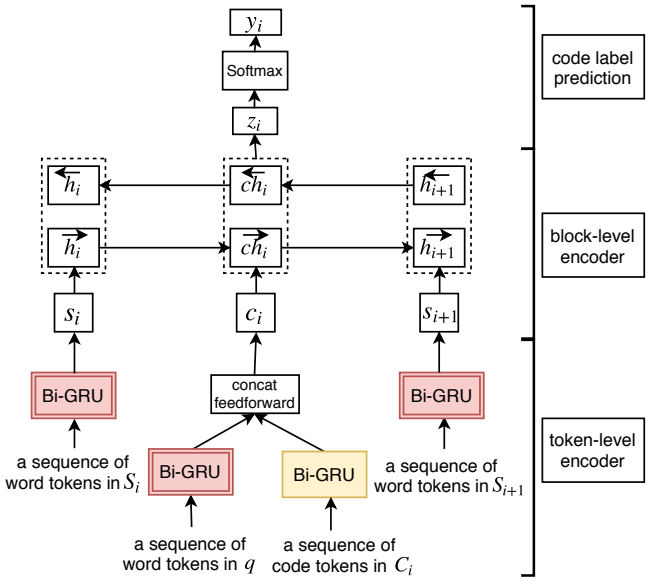


Figure 2: Our Bi-View Hierarchical Neural Network (BiV-HNN). Text block S_i and question q are encoded by a bidirectional GRU-based RNN (Bi-GRU) module and code block C_i is encoded by another Bi-GRU with different parameters.

to learn a new hidden state $\tilde{h}_t$:

$$r = \sigma(W_r[x_t, h_{t-1}] + b_r),$$
$$\tilde{h}_t = \phi(W[x_t, r \odot h_{t-1}] + b),$$

where $[x_t, h_{t-1}]$ is the concatenation of x_t and h_{t-1}, σ and ϕ are the *sigmoid* and *tanh* activation function respectively. W_r, W are two weight matrices in $R^{d_h \times (d_x + d_h)}$ and b_r, b are the biases in R^{d_h}, where d_x, d_h is the dimension of x_t and the hidden state h_{t-1} respectively. Intuitively, if r is close to 0, then the information in h_{t-1} will not be passed into the current step when learning the new hidden state. The GRU also defines an *update gate* u for integrating hidden states h_{t-1} and $\tilde{h}_t$:

$$u = \sigma(W_u[x_t, h_{t-1}] + b_u),$$
$$h_t = uh_{t-1} + (1-u)\tilde{h}_t.$$

When u is closer to 0, h_t contains more information about the current step $\tilde{h}_t$; otherwise, it memorizes more about previous steps. Onwards, we denote the above calculation by $h_t = GRU(x_t, h_{t-1})$ for convenience.

In our work, the bidirectional GRU (i.e., Bi-GRU) contains a forward GRU reading a text block S_i from w_{i1} to w_{iT_i} and a backward GRU which reads from w_{iT_i} to w_{i1}:

$$\overrightarrow{h}_{it} = GRU(x_{it}, \overrightarrow{h}_{i,t-1}), t \in [1, T_i],$$
$$\overleftarrow{h}_{it} = GRU(x_{it}, \overleftarrow{h}_{i,t+1}), t \in [T_i, 1],$$
$$\overrightarrow{h}_{i0} = \overrightarrow{0}, \quad \overleftarrow{h}_{i,T_i+1} = \overleftarrow{0},$$

where the hidden states in both directions are initialized with zero vectors. Since the forward and backward GRU summarize the context information from different perspectives, we concatenate their last hidden states (i.e., $\overrightarrow{h}_{iT_i}$, $\overleftarrow{h}_{i1}$) to represent the meaning of the text block S_i:

$$s_i = [\overrightarrow{h}_{iT_i}, \overleftarrow{h}_{i1}].$$

Code block. Similarly, we employ another Bi-GRU RNN module to learn a vector representation v_c for code block C_i based on its code token sequence. One may directly take this code vector v_c as the token-level representation of a code block. However, since the goal of our model is to decide whether a code snippet answers a certain question, we associate C_i with the question title q to capture their semantic correspondences in the learnt vector representation c_i. Specifically, we first learn the *question vector* v_q by applying the token-level *text* encoder to the word sequence in q. The concatenation of v_q and v_c is then fed into a feedforward *tanh* layer (i.e., "concat feedforward" in Figure 2) for generating c_i:

$$c_i = \phi(W_c[v_q, v_c] + b_c).$$

We will verify the effect of incorporating q in our experiments.

Unlike modeling a code block, we do not associate a text block with question q when learning its representation, because we observed no direct semantic matching between the two. For example, in Figure 1, a text block can hardly match the question by its content. However, as we discussed in Section 1, a text block with patterns like "you can do ..." or "This is one thorough solution ..." can imply that a code solution will be introduced after it. Therefore, we model each text block per se, without incorporating question information.

3.3 Block-level Sequence Encoder

Given the sequence of token-level representations s_i-c_i-s_{i+1}, we use a bidirectional GRU-based RNN to build a block-level sequence encoder and finally obtain the code block representation:

$$\overrightarrow{h}_i = GRU(s_i, \overrightarrow{0}), \overleftarrow{h}_i = GRU(s_i, \overleftarrow{ch}_i),$$
$$\overrightarrow{ch}_i = GRU(c_i, \overrightarrow{h}_i), \overleftarrow{ch}_i = GRU(c_i, \overleftarrow{h}_{i+1}),$$
$$\overrightarrow{h}_{i+1} = GRU(s_{i+1}, \overrightarrow{ch}_i), \overleftarrow{h}_{i+1} = GRU(s_{i+1}, \overleftarrow{0}),$$

where the encoder is initialized with zero vectors (i.e., $\overrightarrow{0}$ and $\overleftarrow{0}$) in both directions. We concatenate the forward state $\overrightarrow{ch}_i$ and the backward state $\overleftarrow{ch}_i$ of the code block as its semantic representation:

$$z_i = [\overrightarrow{ch}_i, \overleftarrow{ch}_i].$$

3.4 Code Label Prediction

The representation z_i of code block C_i is then used for prediction:

$$y_i = \text{softmax}(W_y z_i + b_y),$$

where $y_i = [y_{i0}, y_{i1}]$ represents the probability of predicting C_i to have label 0 or 1 respectively.

We define the loss function using cross entropy [19], which is averaged over all the N code snippets during training:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} p_{i0}\log(y_{i0}) + p_{i1}\log(y_{i1}),$$

where $p_{i0} = 0$ and $p_{i1} = 1$ if the i-th code snippet is manually annotated as a solution; otherwise, $p_{i0} = 1$ and $p_{i1} = 0$.

4 TRADITIONAL CLASSIFIERS WITH FEATURE ENGINEERING

In addition to neural network based models like BiV-HNN, we also explore traditional classifiers like Logistic Regression (LR) [11] and Support Vector Machine (SVM) [10] for our task. Features are manually crafted from both text- and code-based views:

Textual Context. (1) `Token`: The unigrams and bigrams in the context. (2) `FirstToken`: If a sentence starts with phrases like "try this" or "use it", then the following code snippet is very likely to be the solution. Inspired by this idea, we discriminate the first token from others in the context. (3) `Conn`: Boolean features indicating whether a connective word/phrase (e.g., "alternatively") occurs in the context. We used the common connective words and phrases from Penn Discourse Tree Bank [43].

Code Content. (1) `CodeToken`: All code tokens in a code snippet. (2) `CodeClass`: To discriminate code snippets that function and can be considered for learning and pragmatic reuse (i.e., "working code" [22]) from input-output demos, we introduce `CodeClass`, which is the probability of a code snippet being a working code. Specifically, from all the "how-to-do-it" Python questions in SO, we first collected totally 850 code snippets following text blocks such as "output:" and "output is:" as input-output code snippets. We further randomly selected 850 accepted answer posts containing exactly one code snippet and took their code snippets as the working code. We then extracted a set of features like the proportion of numbers and parenthesis and constructed a binary Logistic Regression classifier, which obtains 0.804 accuracy and 0.891 F_1 on a manually labeled testing set. Finally, the trained classifier outputs the probability for each code snippet in Python being a "working code" as the `CodeClass` feature. For SQL, a working code can usually be detected by keywords like "SELECT" and "DELETE", which have been included in the `CodeToken` feature. Thus, we did not design the `CodeClass` feature for it.

There could be other features to incorporate into traditional classifiers. However, coming up with useful features is anything but an easy task. In contrast, neural network models can automatically learn advanced features from raw data and have been broadly and successfully applied in different areas [8, 24, 30, 50, 53]. Therefore, in our work, we choose to design the neural network based model BiV-HNN. We will compare different models in experiments.

5 EXPERIMENTS

In this section, we conduct extensive experiments to compare various models and show the advantages of our proposed BiV-HNN.

5.1 Experimental Setup

Dataset Summarization. Section 2 discussed how we manually annotated question-code pairs for training, validation and testing. Statistics were summarized in Table 1. To evaluate different models, we adopt precision, recall, F_1, and accuracy, which are defined in the same way as in a typical binary classification setting.

Data Preprocessing. We tokenized Python code snippets with best efforts: We first applied Python built-in tokenizer and for code lines that remain untokenized after that, we adopted the "wordpunct_tokenizer" in NLTK toolkit [27] to separate tokens and symbols (e.g., "." and "="). In addition, we detected variables, numbers and strings in a code snippet by traversing its Abstract Syntax Tree (AST) parsed with Python built-in AST parser, and replaced them with special tokens "VAR", "NUMBER" and "STRING" respectively, to alleviate data sparsity. For SQL, we followed [21] to perform the tokenization, which replaced table/column names with placeholder tokens and numbered them to preserve their dependencies. Finally, we collected 4,557 (3,191) word tokens and 6,581 (1,200) code tokens from Python (SQL) training set.

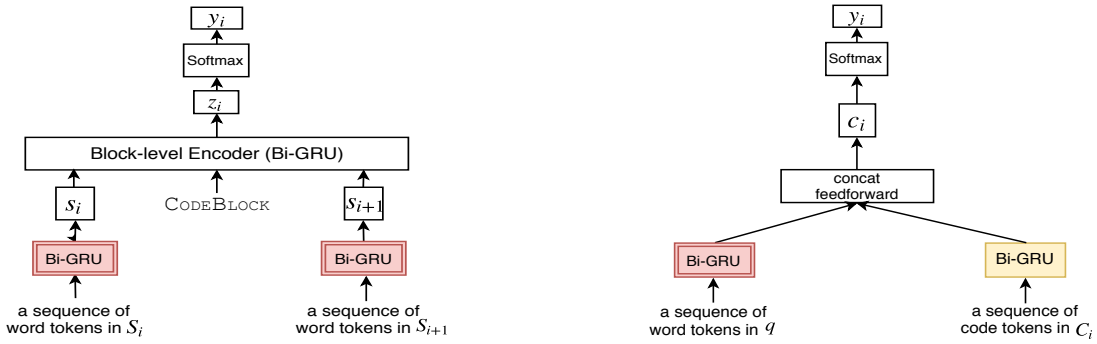
Implementation Details. We used Tensorflow[55] to implement our BiV-HNN and its variants to be introduced in Section 5.2. The embedding size of word and code tokens was set at 150. The embedding vectors were pre-trained using GloVe [42] on all Python or SQL posts in SO. Parameters were randomly initialized following [18]. We started the learning rate at 0.001 and trained neural network models in mini batch of size 100 with the Adam optimizer [23]. The size of the GRU units was chosen from {64, 128} for token-level encoders and from {128, 256} for block-level encoders. Following the convention [20, 21, 29], we selected model parameters based on their performance on validation sets. The Logistic Regression and Support Vector Machine models were implemented with Python Scikit-learn library [41].

5.2 Baselines and Variants of BiV-HNN

Baselines. We compare our proposed model with two commonly used heuristics for collecting QC pairs: (1) *Select-First*: Only treat the first code snippet in an answer post as a solution; (2) *Select-All*: Treat every code snippet in an answer post as a solution and pair each of them with the question. In addition, we compare our model with traditional classifiers like LR and SVM based on hand-crafted features (Section 4).

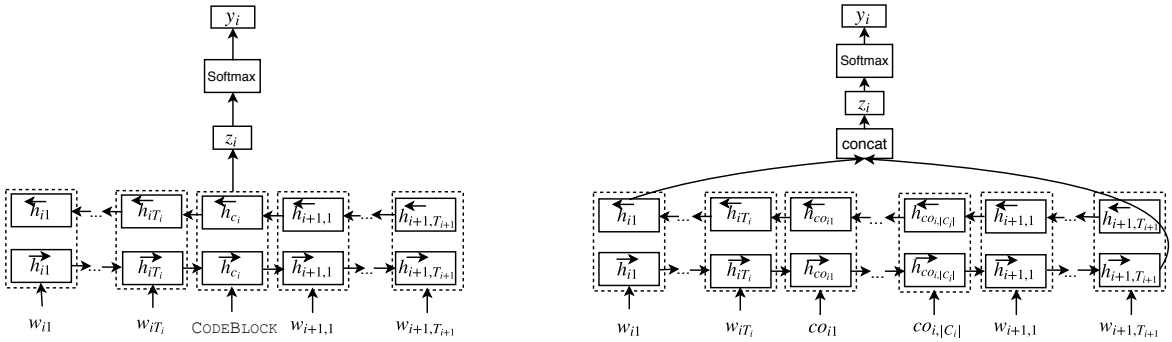
Variants of BiV-HNN. *First*, to evaluate the effectiveness of combining two views (i.e., textual context and code content), we adapt BiV-HNN to consider only one single view: (1) *Text-HNN* (Figure 3a): In this model, we only utilize textual contexts of a code snippet. We mask all code blocks with a special token CodeBlock and represent them with a unified vector. (2) *Code-HNN* (Figure 3b): We only feed the output of the token-level code encoder (i.e., c_i) into the "code label prediction" layer in Section 3, and do not model textual contexts. In addition, to evaluate the effect of question q when encoding a code block, we compare BiV-HNN with *BiV-HNN-nq*, which directly takes the code vector v_c as the code block representation c_i, without associating question q, for further learning. These three models are all input-level variants of BiV-HNN.

Second, to evaluate the hierarchical structure in BiV-HNN, we compare it with "flat" RNN models, which model word and code tokens as a single sequence. The comparison is conducted in both text-only and bi-view settings: (1) *Text-RNN* (Figure 4a): Compared with Text-HNN, we concatenate all words in context blocks S_i and S_{i+1} as well as the unified code vector CodeBlock as a single sequence, i.e., $\{w_{i1}, ..., w_{i,T_i}, \text{CodeBlock}, w_{i+1,1}, ..., w_{i+1,T_{i+1}}\}$, using Bi-GRU RNN. The concatenation of the forward and backward



(a) Text-HNN **(b) Code-HNN**

Figure 3: Single-view variants of BiV-HNN: (a) Text-HNN, without code content; (b) Code-HNN, without contextual text.



(a) Text-RNN **(b) BiV-RNN**

Figure 4: "Flat"-structure variants of BiV-HNN, without differentiating token- and block-level: (a) Text-RNN; (b) BiV-RNN.

hidden states of CodeBlock is considered as its final semantic vector z_i, which is then fed into the code label prediction layer. (2) *BiV-RNN* (Figure 4b): In contrast to BiV-HNN, BiV-RNN models all word and code tokens in S_i-C_i-S_{i+1} as a single sequence, i.e., $\{w_{i1}, ..., w_{iT_i}, co_{i1}, ..., co_{ij}, ..., co_{i,|C_i|}, w_{i+1,1}, ..., w_{i+1,T_{i+1}}\}$, where co_{ij} denotes the j-th token in code C_i and $|C_i|$ is the number of code tokens in C_i. BiV-RNN concatenates the last hidden states in two directions as the final semantic vector z_i for prediction. We also tried directly "flattening" BiV-HNN by concatenating tokens in S_i-q-C_i-S_{i+1}, but observed worse performance, perhaps because transitioning from S_i to question q is less natural.

Finally, at the block level, instead of using an RNN, one may apply a feedforward neural network [47] to the concatenated token-level output $[s_i, c_i, s_{i+1}]$. Specifically, the block-level Bi-GRU in BiV-HNN can be replaced with a one-layer[5] feedforward neural network, denoted as *BiV-HFF*. Intuitively, modeling the three blocks as a sequence is more consistent with the way humans read a post. We will verify this intuition in experiments.

While there could be other variants of our model, the above ones are related to the most critical designs in BiV-HNN. We only show their performance due to space constraints.

[5]For fair comparison, we only use one layer since the Bi-GRU in BiV-HNN only has one hidden layer.

5.3 Results

Our experimental results in Table 2 show the effectiveness of our BiV-HNN. On both datasets, BiV-HNN substantially outperforms heuristic baselines Select-First and Select-All by more than **15%** in F_1 and accuracy. This demonstrates that our model can collect QC pairs with much higher quality than heuristic methods used in existing research. In addition, when compared with LR and SVM, BiV-HNN achieves 7%∼9% higher F_1 and accuracy on Python dataset, and 3%∼5% better F_1 and accuracy on SQL dataset. The gain on SQL data is relatively smaller, probably because interpreting SQL programs is a relatively easier task, implied by the observation that both simple classifiers and BiV-HNN can have around 85% F1.

Results in Table 3 show the effect of key components in BiV-HNN in comparison with alternatives. Due to space constraints, we do not show the accuracy of each model, which has roughly the same pattern as F_1. We have made the following observations: (1) *Single-view variants*. BiV-HNN outperforms Text-HNN and Code-HNN by a large margin on both datasets, showing that both views are critical for our task. In particular, by incorporating code content information, BiV-HNN is able to improve Text-HNN by 7% on Python dataset and around 5% on SQL dataset in F_1. (2) *No-query variant*. On Python dataset, the integration of the question information in BiV-HNN brings 3% F_1 improvements over BiV-HNN-nq,

Model	Python Testing Set				SQL Testing Set			
	Precision	Recall	F_1	Accuracy	Precision	Recall	F_1	Accuracy
Heuristics Baselines								
Select-First	0.676	0.551	0.607	0.663	0.755	0.517	0.613	0.620
Select-All	0.472	**1.000**	0.642	0.472	0.583	**1.000**	0.737	0.583
Classifiers based on simple features								
Logistic Regression	0.801	0.733	0.766	0.788	0.843	0.849	0.846	0.820
Support Vector Machine	0.701	0.813	0.753	0.748	0.843	0.858	0.850	0.824
BiV-HNN	**0.808**	0.876	**0.841**	**0.843**	**0.872**	0.903	**0.888**	**0.867**

Table 2: Comparison of BiV-HNN and baseline methods.

which shows the effectiveness of associating the question with the code snippet for identifying code answers. For SQL dataset, adding the question gives no obvious benefit, possibly because the code content in each SQL program already carries critical information for making a prediction (e.g., a SQL program containing the command keyword "SELECT" is very likely to be a solution to the given question, regardless of the question content). (3) *"Flat"-structure variants.* On both datasets, the hierarchical structure leads to 1%∼2% improvements against the "flat" structure in both bi-view (BiV-HNN vs. BiV-RNN) and single-view setting (Text-HNN vs. Text-RNN). (4) *Non-sequence variant.* On Python dataset, BiV-HNN outperforms BiV-HFF by around 2%, showing the block-level Bi-GRU is preferable over feedforward neural networks. The two models get roughly the same performance on SQL, probably because our task is easier in SQL domain than in Python domain as we mentioned earlier.

In summary, our BiV-HNN is much more effective than widely-adopted heuristic baselines and traditional classifiers. The key components in BiV-HNN, such as bi-view inputs, hierarchical structure and block-level sequence encoding, are also empirically justified.

Error Analysis. There are a variety of non-solution roles that a code snippet can play, such as being only one step of a multi-step solution, an input-output example, etc. We observe that more than half of the wrong predictions were false positives (i.e., predicting a non-solution code snippet as a solution), correcting which usually requires integrating information from the entire answer post. For example, when a code snippet is the first step of a multi-step solution, BiV-HNN may mistakenly take it as a complete and standalone solution, since BiV-HNN does not simultaneously take into account follow-up code snippets and their context to make predictions. In addition, BiV-HNN may make mistakes when a correct prediction requires a close examination of the *content* of a question post (besides its title). Exploring these directions in the future may lead to further improved model performance on this task.

Model Combination. When experimenting with the single-view variants of BiV-HNN, i.e., Text-HNN and Code-HNN, we observed that the three models complement each other in making accurate predictions. For example, on Python validation set, around 70% mistakes made by Text-HNN or Code-HNN can be corrected by considering predictions from the other two models. Although BiV-HNN is built based on both text- and code-based views, 60% of its wrong predictions can be remedied by Text-HNN and Code-HNN. The same pattern was also observed on SQL dataset.

Therefore, we further tested the effect of combining the three models via a simple heuristic: The label of a code snippet is predicted only when the three models agree on it. Using this heuristic, 69.2% code blocks on the annotated Python testing set are labeled with 0.916 F_1 and 0.911 accuracy. Similarly, on SQL testing set, 78.7% code blocks are labeled with 0.943 F_1 and 0.926 accuracy. The combined model further improves BiV-HNN by around 6% while still being able to label a large portion of the code snippets. Thus, we apply this combined model to those SO answer posts that are not manually annotated yet to obtain large-scale QC pairs, to be discussed next.

Model	Python Testing Set			SQL Testing Set		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
Single-view Variants						
Text-HNN	0.723	0.826	0.771	0.798	0.887	0.840
Code-HNN	0.770	0.859	0.812	0.848	0.854	0.851
No-query Variant						
BiV-HNN-nq	0.802	0.818	0.810	**0.883**	0.892	0.887
"Flat"-structure Variants						
Text-RNN	0.693	0.824	0.753	0.773	0.894	0.829
BiV-RNN	0.760	**0.887**	0.819	0.869	0.880	0.875
Non-sequence Variant						
BiV-HFF	0.787	0.859	0.822	0.845	**0.939**	**0.889**
BiV-HNN	**0.808**	0.876	**0.841**	0.872	0.903	0.888

Table 3: Comparison of BiV-HNN and its variants.

6 STAQC: A SYSTEMATICALLY MINED DATASET OF QUESTION-CODE PAIRS

In this section, we present StaQC (Stack Overflow Question-Code pairs), a large-scale and diverse set of question-code pairs automatically mined using our framework. Under various case studies, we demonstrate that StaQC can greatly help tasks aiming to associate natural language with programming language.

6.1 Statistics of StaQC

In Section 5, we showed that a combination of BiV-HNN and its variants can reliably identify standalone code solutions with > 90% F_1 and accuracy from a large portion of the testing set. Thus we applied this combined model to all unlabeled multi-code answer posts that correspond to "how-to-do-it" questions in Python and SQL domain, and finally collected **60,083** and **41,826** question-code pairs respectively. Additionally, there are 85,294 Python and 75,637 SQL "how-to-do-it" questions whose answer post contains exactly one code snippet. For them, as in [21], we paired the question title

	# of QC pairs	Question		Code	
		Average length	# of tokens	Average length	# of tokens
Python	**147,546**	9	17,635	86	137,123
SQL	**119,519**	9	9,920	60	21,143

Table 4: Statistics of StaQC.

Question: "*How to limit a number to be within a specified range? (Python)*"
Code answers:

```
def clamp(n, minn, maxn):
  return max(min(maxn, n), minn)
```
(1st)

```
clamp = lambda n, minn, maxn: max(min(maxn, n), minn)
```
(2nd)

```
def clamp(n, minn, maxn):
  if n < minn:
    return minn
  elif n > maxn:
    return maxn
  else:
    return n
```
(5th)

```
n = minn if n < minn else maxn if n > maxn else n
```
(4th)

Figure 5: StaQC contains four alternative code solutions to question "*How to limit a number to be within a specified range? (Python)*" [38] whose answer post contains five code snippets. The number at the bottom right denotes the position of each code snippet in the answer post.

with the one code snippet as a question-code pair. Together with 2,169 and 2,056 manually annotated QC pairs with label "1" for each domain (Table 1), we collected a dataset of **147,546** Python and **119,519** SQL QC pairs, named as StaQC. Table 4 shows its statistics.

Note that we can continue to expand StaQC with minimal efforts, since it is automatically mined by our framework, and more and more posts will be created in SO as time goes by. QC pairs in other programming languages can also be mined similarly to further enrich StaQC beyond Python and SQL domain.

6.2 Diversity of StaQC

Besides the large scale, StaQC also enjoys great diversity in the sense that it contains multiple textual descriptions for semantically similar code snippets and multiple code solutions to a question. For example, considering question "*How to limit a number to be within a specified range? (Python)*" whose answer post contains five code snippets (Figure 5), our framework is able to correctly mine four alternative code answers. Heuristic methods may either miss some of them or mistakenly include a false solution (i.e., the 3rd code snippet). Therefore, our framework is able to obtain more alternative solutions for the same question more accurately. Moreover, Figure 6 shows two question-code pairs included in StaQC, which we easily located by comparing code solutions of *relevant* questions in SO (i.e., questions manually linked by SO users). Note that the two code snippets have a very similar functionality but two different text descriptions.

Figure 5 and 6 show that StaQC is highly diverse and rich in surface variation. Such a dataset is beneficial for model development. Intuitively, when certain data patterns are not observed in the training phase, a model is less capable to predict them during testing. StaQC can alleviate this issue by enabling a model to learn from alternative code solutions to the same question or from different

Question A: "*How to find a gap in range in SQL*"

```
SELECT id + 1
FROM test mo
WHERE NOT EXISTS
(
  SELECT NULL
  FROM test mi
  WHERE mi.id = mo.id + 1
  ) and mo.id> 100
ORDER BY
        id
LIMIT 1
```

Question B: "*How do I find a "gap" in running counter with SQL?*"

```
SELECT id + 1
FROM mytable mo
WHERE NOT EXISTS
(
  SELECT NULL
  FROM mytable mi
  WHERE mi.id = mo.id + 1
  )
ORDER BY
        id
LIMIT 1
```

Figure 6: StaQC has different text descriptions, e.g., "*How to find a gap in range in SQL*" [37] and "*How do I find a "gap" in running counter with SQL?*" [36], for two code snippets bearing a similar functionality.

text descriptions to similar code snippets. Next we demonstrate this benefit using an exemplar downstream task.

6.3 Usage Demo of StaQC on Code Retrieval

To further demonstrate the usage of StaQC, we employ it to train a deep learning model for the code retrieval task [2, 21, 22]. *Given a natural language description and a set of code snippet candidates, the task is to retrieve code snippets that can match the description.* In particular, an effective model should rank matched code snippets as high as possible. Models are evaluated by Mean Reciprocal Rank (MRR) [58]. In [21], the authors proposed a neural network based model, *CODE-NN*, which outputs a matching score between a natural language question and a code snippet. We choose CODE-NN as it is one of the state of the arts for code retrieval and improved previous work by a large margin. For training, the authors collected around 25,870 SQL QC pairs from answer posts containing exactly one code snippet (which is paired with the question title). They manually annotated two datasets DEV and EVAL for choosing the best model parameters and for final evaluation respectively, both containing around 100 QC pairs. The final evaluation is conducted in 20 runs. In each run, for every QC pair in DEV or EVAL, [21] randomly selected 49 code snippets from SO as non-answer candidates, and ranked all 50 code snippets based on their scores output by CODE-NN. The averaged MRR is computed as the final result.

Improved Retrieval Performance. We first trained CODE-NN using the *original* training set in [21]. We denote this setting as *CODE-NN (original)*. Then we used StaQC to upgrade the training data in two most straightforward ways: (1) We directly took all the 119,519 SQL QC pairs in StaQC to train CODE-NN, denoted as *CODE-NN (StaQC)*. (2) To emphasize the effect of our framework, we just added the 41,826 QC pairs, automatically mined from SO multi-code answer posts, to the original training set and retrained the model, which is denoted as *CODE-NN (original + StaQC-multi)*. In both (1) and (2), questions and code snippets occurring in the DEV/EVAL set were removed from training.

In all three settings, we used the same DEV/EVAL set and the same hyper-parameters as in [21] except the dropout rate, which was chosen from {0.5, 0.7} for each model to obtain better performance. Like [21], we decayed the learning rate in each epoch and terminated the training when it was lower than 0.001. The best

Model Setting	MRR
CODE-NN (original)	0.51 ± 0.02
CODE-NN (StaQC)*	**0.57 ± 0.02**
CODE-NN (original + StaQC-multi)*	0.54 ± 0.02

Table 5: Performance of CODE-NN [21] on code retrieval, with and without StaQC for training. *denotes statistically significant w.r.t. *CODE-NN (original)* under one-tailed Student's t-test ($p < 0.05$).

model was selected as the one achieving the highest average MRR on DEV set. When using this strategy, we observed better results on the EVAL set than those reported in [21] (around 0.44).

Table 5 shows the average MRR score and standard deviation of each model on EVAL set. We can see that directly using StaQC for training leads to a substantial 6% improvement over using the original dataset in [21]. By adding QC pairs we mined from multi-code posts to the original training data, CODE-NN can be significantly improved by 3%. Note that the performance gains shown here are still conservative, since we adopted the same hyper-parameters and a small evaluation set, in order to see the direct impact of StaQC. Using more challenging evaluation sets and by conducting systematic hyper-parameter selection, we expect models trained on StaQC to be more advantageous. StaQC can also be used to train other code retrieval models besides CODE-NN, as well as models for other related tasks like code generation or annotation.

7 DISCUSSION AND FUTURE WORK

Besides boosting relevant tasks using StaQC, future work includes: (1) We currently only consider a code snippet to be a standalone solution or not. In many cases, code snippets in an answer post serve as multiple steps and should be merged to form a complete solution [39]. This is a more challenging task and we leave it to the future. (2) In our experiments, we combined BiV-HNN and its two variants using a simple heuristic to achieve better performance. In the future, one can also use StaQC to retrain the three models, similar to self-training [34], or jointly train the three models in a tri-training framework [63]. (3) One may also employ Convolutional Neural Networks [1, 24, 49], which have shown great power on representation learning, to encode text and code blocks. Moreover, we can consider encoders similar to [31, 33] for capturing the intrinsic structure of programming language.

8 RELATED WORK

***Language + Code* Tasks and Datasets**. Tasks that map between natural language and programming language, referred to *Language + Code* tasks here, such as code annotation and code retrieval/generation, have been popularly studied in recent years [2, 15, 21, 22, 26, 35, 45, 57, 64]. In order to train more advanced yet data-hungry models, researchers have collected data either automatically from online communities [2, 5, 21, 22, 26, 45, 57, 64] or with human intervention [16, 35]. Like our work, [2, 21, 57, 64] utilized SO to collect data. Particularly, [2] merges code snippets in its answer post as the target source code and pair it with the question title. [21] only employs accepted answer posts containing exactly one code snippet. Other interesting datasets include ~19K <English pseudo-code, Python code snippet> pairs manually annotated by [35], and ~114K pairs of Python functions and their documentation strings heuristically collected by [5] from GitHub [17]. Unlike their work, we systematically mine high-quality question-code pairs from SO using advanced machine learning models. Our mined dataset StaQC, the largest to date of around 148K Python and 120K SQL question-code pairs, has been shown to be a better resource. Moreover, StaQC is easily expandable in terms of both scale and programming language types.

Recurrent Neural Networks for Sequential Data. Recurrent Neural Networks have shown great success in various natural language tasks [4, 8, 20, 29]. In an RNN, terms are modeled sequentially without discrimination. Recently, in order to handle information at different levels, [25, 48, 54, 60] stack multiple RNNs into a hierarchical structure. For example, [60] incorporates the attention mechanism in a hierarchical RNN model to pick up important words and sentences. Their model finally aggregates all sentence vectors to learn the document representation. In comparison, we utilize the hierarchical structure to first learn the semantic meaning of each block individually, and then predict the label of a code snippet by combining two views: textual context and programming content.

Mining Stack Overflow. Stack Overflow has been the focus of the Mining Software Repositories (MSR) challenge for years [3, 62]. A lot of work [12–14, 32, 56, 59] have been done on exploring the categories of questions, mining source codes, etc. We follow [12, 13, 32] to categorize SO questions into 5 classes but only focus on the "how-to-do-it" type (Section 2). [14, 59] analyzes the quality of code snippets (e.g., readability) or explores "usable" code snippets that could be parsed, compiled and run. Different from their work, we are interested in finding standalone code solutions, which are not necessarily directly parsable, compilable or runnable, but can be semantically paired with questions. To the best of our knowledge, we are the first to study the problem of systematically mining high-quality question-code pairs.

9 CONCLUSION

This paper explores systematically mining question-code pairs from Stack Overflow, in contrast to heuristically collecting them. We focus on the "how-to-do-it" questions since their answers are more likely to be code solutions. We present the largest-to-date dataset of diversified question-code pairs in Python and SQL domain (StaQC), systematically collected by our framework. StaQC can greatly help downstream tasks aiming to associate natural language with programming language. We will release it together with our source code for future research.

ACKNOWLEDGMENTS

This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, Fujitsu gift grant, DARPA contract FA8750-13-2-0019, the University of Washington WRF/Cable Professorship, Ohio Supercomputer Center [7], and NSF Grant CNS-1513120. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

REFERENCES

[1] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In *ICML*. 2091–2100.

[2] Miltos Allamanis, Daniel Tarlow, Andrew Gordon, and Yi Wei. 2015. Bimodal modelling of source code and natural language. In *ICML*. 2123–2132.

[3] Alberto Bacchelli. 2013. Mining Challenge 2013: Stack Overflow. In *The 10th Working Conference on Mining Software Repositories*. to appear.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR* abs/1409.0473 (2014). arXiv:1409.0473 http://arxiv.org/abs/1409.0473

[5] Antonio Valerio Miceli Barone and Rico Sennrich. 2017. A parallel corpus of Python functions and documentation strings for automated code documentation and code generation. *arXiv preprint arXiv:1707.02275* (2017).

[6] Brock Angus Campbell and Christoph Treude. 2017. NLP2Code: Code Snippet Content Assist via Natural Language Tasks. *arXiv preprint arXiv:1701.05648* (2017).

[7] Ohio Supercomputer Center. 1987. Ohio Supercomputer Center. http://osc.edu/ark:/19495/f5s1ph73. (1987).

[8] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *EMNLP*. Association for Computational Linguistics, Doha, Qatar, 1724–1734.

[9] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[10] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[11] David R Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)* (1958), 215–242.

[12] Lucas BL de Souza, Eduardo C Campos, and Marcelo de A Maia. 2014. Ranking crowd knowledge to assist software development. In *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 72–82.

[13] Fernanda Madeiral Delfim, Klérisson VR Paixão, Damien Cassou, and Marcelo de Almeida Maia. 2016. Redocumenting APIs with crowd knowledge: a coverage analysis based on question types. *Journal of the Brazilian Computer Society* 22, 1 (2016), 9.

[14] Maarten Duijn, Adam Kučera, and Alberto Bacchelli. 2015. Quality questions need quality code: classifying code fragments on stack overflow. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 410–413.

[15] Alessandra Giordani and Alessandro Moschitti. 2009. Semantic mapping between natural language questions and SQL queries via syntactic pairing. In *International Conference on Application of Natural Language to Information Systems*. Springer, 207–221.

[16] Alessandra Giordani and Alessandro Moschitti. 2010. Corpora for Automatically Learning to Map Natural Language Questions into SQL Queries.. In *LREC*.

[17] GitHub. 2017. GitHub. (2017). https://github.com/

[18] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 249–256.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[20] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*. 1693–1701.

[21] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *ACL*, Vol. 1. 2073–2083.

[22] Iman Keivanloo, Juergen Rilling, and Ying Zou. 2014. Spotting working code examples. In *ICSE*. ACM, 664–675.

[23] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105.

[25] Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. 2015. A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057* (2015).

[26] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744* (2016).

[27] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1 (ETMTNLP '02)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 63–70. https://doi.org/10.3115/1118108.1118117

[28] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes. *arXiv preprint arXiv:1704.04856* (2017).

[29] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *EMNLP*.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[31] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *AAAI*.

[32] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example?: A study of programming Q&A in StackOverflow. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 25–34.

[33] Anh Tuan Nguyen and Tien N Nguyen. 2015. Graph-based statistical language model for code. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 858–868.

[34] Kamal Nigam and Rayid Ghani. 2000. Analyzing the effectiveness and applicability of co-training. In *CIKM*. ACM, 86–93.

[35] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 574–584.

[36] Stack Overflow. 2017. How do I find a "gap" in running counter with SQL? (2017). https://stackoverflow.com/a/1312137/4941215

[37] Stack Overflow. 2017. How to find a gap in range in SQL. (2017). https://stackoverflow.com/a/17782635/4941215

[38] Stack Overflow. 2017. How to limit a number to be within a specified range? (Python). (2017). https://stackoverflow.com/a/5996949/4941215

[39] Stack Overflow. 2017. Splitting a dataframe based on column values. (2017). https://stackoverflow.com/a/33973304/4941215

[40] Stack Overflow. 2017. Stack Overflow. (2017). https://stackoverflow.com/

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[42] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global Vectors for Word Representation.. In *EMNLP*, Vol. 14. 1532–1543.

[43] Rashmi Prasad, Nikhil Dinesh, Alan Lee, Eleni Miltsakaki, Livio Robaldo, Aravind Joshi, and Bonnie Webber. 2008. The Penn Discourse TreeBank 2.0. In *In Proceedings of LREC*.

[44] Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract Syntax Networks for Code Generation and Semantic Parsing. In *ACL*.

[45] Mukund Raghothaman, Yi Wei, and Youssef Hamadi. 2016. SWIM: synthesizing what I mean: code search and idiomatic snippet synthesis. In *ICSE*. ACM, 357–367.

[46] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. In *NIPS*. 3567–3575.

[47] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.

[48] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. 2016. Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models.. In *AAAI*. 3776–3784.

[49] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*. ACM, 101–110.

[50] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[51] Inc Stack Exchange. 2017. Stack Exchange Data Dump. (2017). https://archive.org/details/stackexchange

[52] Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, and Michael Gamon. 2017. Building Natural Language Interfaces to Web APIs, In CIKM.

[53] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. 2013. Deep neural networks for object detection. In *NIPS*. 2553–2561.

[54] Duyu Tang, Bing Qin, and Ting Liu. 2015. Document Modeling with Gated Recurrent Neural Network for Sentiment Classification.. In *EMNLP*. 1422–1432.

[55] TensorFlow. 2017. TensorFlow. (2017). https://www.tensorflow.org/

[56] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. 2011. How do programmers ask and answer questions on the web?: Nier track. In *ICSE*. IEEE, 804–807.

[57] Venkatesh Vinayakarao, Anita Sarma, Rahul Purandare, Shuktika Jain, and Saumya Jain. 2017. Anne: Improving source code search using entity retrieval approach. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 211–220.

[58] Ellen M Voorhees et al. 1999. The TREC-8 Question Answering Track Report.. In *Trec*, Vol. 99. 77–82.

[59] Di Yang, Aftab Hussain, and Cristina Videira Lopes. 2016. From query to usable code: an analysis of stack overflow code snippets. In *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM, 391–402.

[60] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT*. 1480–1489.

[61] Pengcheng Yin and Graham Neubig. 2017. A Syntactic Neural Model for General-Purpose Code Generation. In *ACL*. Vancouver, Canada.

[62] Annie T. T. Ying. 2015. Mining Challenge 2015: Comparing and combining different information sources on the Stack Overflow data set. In *The 12th Working Conference on Mining Software Repositories*. to appear.

[63] Zhi-Hua Zhou and Ming Li. 2005. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on knowledge and Data Engineering* 17, 11 (2005), 1529–1541.

[64] Meital Zilberstein and Eran Yahav. 2016. Leveraging a corpus of natural language descriptions for program similarity. In *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. ACM, 197–211.

Interactive Semantic Parsing for If-Then Recipes via Hierarchical Reinforcement Learning

Ziyu Yao[⋆] **Xiujun Li**[†§] **Jianfeng Gao**[†] **Brian Sadler**[‡] **Huan Sun**[⋆]
[⋆]The Ohio State University [§]University of Washington
[†]Microsoft Research AI [‡]U.S. Army Research Lab
{yao.470,sun.397}@osu.edu {xiul,jfgao}@microsoft.com
brian.m.sadler6.civ@mail.mil

Abstract

Given a text description, most existing semantic parsers synthesize a program in one shot. However, it is quite challenging to produce a correct program solely based on the description, which in reality is often ambiguous or incomplete. In this paper, we investigate *interactive semantic parsing*, where the agent can ask the user clarification questions to resolve ambiguities via a multi-turn dialogue, on an important type of programs called "If-Then recipes." We develop a hierarchical reinforcement learning (HRL) based agent that significantly improves the parsing performance with minimal questions to the user. Results under both simulation and human evaluation show that our agent substantially outperforms non-interactive semantic parsers and rule-based agents.[1]

1 Introduction

Semantic parsing aims to map natural language to formal domain-specific meaning representations, such as knowledge base or database queries (Berant et al. 2013; Dong and Lapata 2016; Zhong, Xiong, and Socher 2017; Gao, Galley, and Li 2018), API calls (Campagna et al. 2017; Su et al. 2017) and general-purpose code snippets (Yin and Neubig 2017; Rabinovich, Stern, and Klein 2017). In this work, we focus on semantic parsing for synthesizing a simple yet important type of conditional statements called *If-Then recipes* (or *If-Then programs*), based on a natural language description (Quirk, Mooney, and Galley 2015; Beltagy and Quirk 2016; Liu et al. 2016; Yin and Neubig 2017; Chaurasia and Mooney 2017). For example, the description "Create a link note on Evernote for my liked tweets" should be parsed into an If-Then recipe with 4 components: *trigger channel* `Twitter`, *trigger function* `New liked tweet by you`, *action channel* `Evernote`, and *action function* `Create a link note`. On the one hand, If-Then recipes allow users to perform a large variety of tasks such as home security ("text me if the door is not locked"). On the other hand, developing intelligent agents that can automatically parse these recipes is an important step towards complex natural language programming (Quirk, Mooney, and Galley 2015).

[1]All source code and documentations are available at https://github.com/LittleYUYU/Interactive-Semantic-Parsing.

User: "record to evernote" **Ground-truth recipe**: [tc: `Twitter`, tf: `New liked tweet by you`, ac: `Evernote`, af: `Create a link note`] **(Liu et al. 2016)**: [tc: `Phone Call`, tf: `Leave IFTTT any voicemail`, ac: `Evernote`, af: `Append to note`]
User: "record to evernote" **HRL-fixedOrder agent**: "Which channel triggers the action?" **User**: "twitter" **HRL-fixedOrder agent**: "Which event triggers the action?" **User**: "If I like a tweet" **HRL-fixedOrder agent**: "Which event results from the trigger?" **User**: "Create a note with link" **Agent Prediction**: [tc: `Twitter`, tf: `New liked tweet by you`, ac: `Evernote`, af: `Create a link note`]
User: "record to evernote" **HRL agent**: "Which event triggers the action?" **User**: "If I like a tweet" **HRL agent**: "Which event results from the trigger?" **User**: "Create a note with link" **Agent Prediction**: [tc: `Twitter`, tf: `New liked tweet by you`, ac: `Evernote`, af: `Create a link note`]

Table 1: Semantic parsers on an ambiguous description: The state-of-the-art non-interactive model (Liu et al. 2016) cannot correctly parse the recipe while our two HRL-based interactive agents can. Particularly, by coordinating the subtask order, the HRL agent asks fewer questions than the HRL-fixedOrder agent (tc: trigger channel, tf: trigger function, ac: action channel: af: action function).

Most previous work translates a natural language description to an If-Then recipe *in one turn*: The user gives a recipe description and the system predicts the 4 components. However, in reality, a natural language description can be very noisy and ambiguous, and may not contain enough information. For simplicity, we refer to this problem as *description ambiguity*. In fact, in the widely used If-Then evaluation dataset (Quirk, Mooney, and Galley 2015), **80%** of the descriptions are ambiguous. As shown in Table 1, the description "record to evernote" is paired with the same ground-truth recipe as in the first example, but even humans cannot tell what the "record" refers to (i.e., trigger channel/function) and what kind of note to create on Evernote

(i.e., action function). Therefore, it is quite challenging, if not impossible, to produce a correct program in one shot merely based on an ambiguous description.

Driven by this observation, we investigate *interactive semantic parsing*, where an intelligent agent (e.g., the two HRL-based agents in Table 1) strives to improve the parsing accuracy by asking clarification questions. Two key challenges are addressed: (1) Lack of supervision on when to ask a question. To date, there is no large-scale annotated dataset on whether and when an agent should ask a question during parsing. The only feedback an agent can obtain is whether or not a synthesized program is correct. (2) How to improve the parsing accuracy with a minimal number of questions? To guarantee a good user experience, the agent should only ask "necessary" questions and learn from human interactions over time.

Previous work (Chaurasia and Mooney 2017) developed rule-based agents to interactively predict the 4 components of an If-Then recipe. These agents decide to ask a question when the prediction probability of a recipe component is lower than a predefined threshold. However, such rule-based agents are not trained in an optimization framework to simultaneously improve the parsing accuracy and reduce the number of questions.

We address these challenges via a Hierarchical Reinforcement Learning (HRL) approach. We formulate the interactive semantic parsing in the framework of *options* over Markov Decision Processes (MDPs) (Sutton, Precup, and Singh 1999), where the task of synthesizing an If-Then recipe is naturally decomposed into 4 *subtasks* or *options* (i.e., predicting trigger/action channel/function). In particular, we propose an HRL agent with a hierarchical policy: A high-level policy decides the order of the subtasks to work on, and a low-level policy for each subtask guides its completion by deciding whether to (continue to) ask a clarification question or to predict the subtask component. We train the policies to maximize the parsing accuracy and minimize the number of questions with the rewarding mechanism, where the only supervision (reward signal) is whether or not a predicted component is correct.

Compared with the approach of solving the entire task with one flat policy (Mnih et al. 2015), HRL takes advantage of the naturally defined "4-subtask" structure. Such design also allows the agent to focus on different parts of a recipe description for each subtask, as emphasized in (Liu et al. 2016), and endows each low-level policy with a reduced state-action space to simplify the learning. On the other hand, the high-level policy optimizes the subtask order by taking into account both the recipe description and user responses. As shown in Table 1, the HRL agent learns to ask about the trigger function first, to which the user response (i.e., "tweet") implies the trigger channel. This mechanism leads to fewer questions than the HRL-fixedOrder agent, which executes subtasks in the fixed order of "tc-tf-ac-af".

Experimental results under both simulation and human evaluation show that our HRL agent can obtain a significantly better accuracy while asking fewer questions than rule-based agents like (Chaurasia and Mooney 2017). In addition, we show the effectiveness of the high-level policy on reducing the number of questions. Our agent tends to predict functions before channels, which is different from most existing works that either assume independence among subtasks (Chaurasia and Mooney 2017) or predict channels before functions (Beltagy and Quirk 2016; Dong and Lapata 2016; Yin and Neubig 2017).

2 Background

If-Then recipes allow people to manage a variety of web services or physical devices and automate event-driven tasks. We focus on recipes from IFTTT.com, where a recipe has 4 components: *trigger channel*, *trigger function*, *action channel*, and *action function*. There are a variety of channels, like `GMail` and `Facebook`, representing entities such as web applications and IFTTT-provided services. Each channel has a set of functions representing events (i.e., trigger functions) or action executions (i.e., action functions). In one recipe, there could be exactly one value for trigger/action channel/function.

Following (Liu et al. 2016; Chaurasia and Mooney 2017), we decompose the task of parsing a natural language description into four subtasks, i.e., predicting trigger/action channel/function in a recipe. We have made two key observations about real-life recipe descriptions and existing semantic parsing work: (1) Around 80% recipe descriptions are ambiguous or contain incomplete information, according to the human annotations provided by Quirk, Mooney, and Galley (2015), which makes it extremely difficult to synthesize a recipe in one turn (see the prediction result of (Liu et al. 2016) in Table 1). Therefore, (Liu et al. 2016; Chaurasia and Mooney 2017; Yin and Neubig 2017) focus on the unambiguous 20% recipes for evaluation. (2) Previous work assumes either independence (Chaurasia and Mooney 2017) or heuristic dependencies among the 4 subtasks. In particular, Liu et al. (2016) assumes that functions should be predicted before channels since a channel can be derived from the function prediction, while (Beltagy and Quirk 2016; Dong and Lapata 2016; Yin and Neubig 2017) assume that channels should be predicted before functions and triggers before actions.

Given these observations, we propose an *interactive semantic parser* that can ask users for clarification to make more accurate predictions. Moreover, we abandon the inter-subtask (in)dependence assumptions used in previous work. Our agent learns to optimize the subtask order for each recipe to save questions.

3 Interactive Semantic Parser

3.1 Framework Overview

Given a recipe description, four components need to be predicted. Thus, the semantic parsing task can be naturally decomposed into four subtasks $\mathcal{G} = \{st_1, st_2, st_3, st_4\}$, standing for predicting the trigger channel (st_1), trigger function (st_2), action channel (st_3) and action function (st_4), respectively. We aim at an agent that can decide the order of subtasks for each parsing task, and only moves to the next subtask when the current one has been completed. We formulate this as a hierarchical decision making problem based on

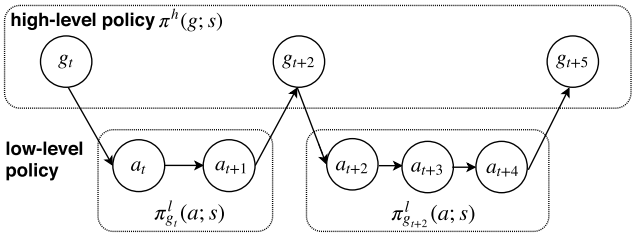


Figure 1: Hierarchical policy: The high-level policy chooses a subtask to work on while the low-level policy decides to predict the subtask or ask the user at each time step.

the framework of *options* over Markov Decision Processes (MDPs) (Sutton, Precup, and Singh 1999).

Specifically, the agent uses a hierarchical policy consisting of two levels of policies operating at different time scales. The high-level policy selects the next subtask (or *option*) to work on, which can be viewed as operating on a Semi-MDP (Sutton, Precup, and Singh 1999). The low-level policy selects primitive actions (i.e., predicting a component value or querying the user) to complete the selected subtask. As elaborated in Section 3.2, we adopt 4 low-level policies, each for one subtask.

Figure 1 shows the process. At an *eligible* time step t (i.e., *at the beginning of a parsing task or when a subtask terminates*), the high-level policy $\pi^h(g; s_t)$ receives a state s_t and selects a subtask $g_t \in \mathcal{G}$ to work on. Then the low-level policy $\pi^l_{g_t}(a; s_t)$ for this subtask chooses an action $a_t \in \mathcal{A}_{g_t}$. By taking this action, the agent receives a low-level reward $r^l_{g_t}(s_t, a_t)$. For the next N time steps (e.g., $N = 2$ in Figure 1), the agent will work on the same subtask g_t until it terminates (i.e., when either the agent has predicted the corresponding component or the agent has interacted with the user for Max_Lobal_Turn turns). The agent will receive a high-level reward $r^h(s_t, g_t)$ for this subtask completion, then select the next subtask g_{t+N} and repeat the above procedure until the entire task terminates (i.e., when either all four components are predicted or the agent has worked on Max_Global_Turn subtasks).

States. A state s tracks 9 items during the course of interactive parsing:

- The initial recipe description $\mathcal{I}$.
- The boolean indicator b_i $(i = 1, 2, 3, 4)$ showing whether subtask st_i has been predicted.
- The received user answer d_i $(i = 1, 2, 3, 4)$ for subtask st_i, respectively.

For each subtask st_i, we learn a *low-level state vector* $s^l_{st_i}$ to summarize state information of this subtask for low-level policy $\pi^l_{st_i}$ to select the next action. Similarly, a *high-level state vector* s^h is learned to present a summary of the entire state, consisting of the 4 low-level state vectors and other state information, for high-level policy π^h to choose the next subtasks. Section 3.2 details how states are represented.

Actions. The action space for the high-level policy is $\mathcal{G} = \{st_1, st_2, st_3, st_4\}$, where each action denotes one subtask mentioned earlier. The action space of subtask g is $\mathcal{A}_g = \mathcal{V}_g \cup \{\text{AskUser}\}$, where $\mathcal{V}_g$ is the set of available component values for subtask g (e.g., all trigger channels for subtask st_1), meaning that the agent can either predict a component or ask the user a clarification question.

Rewards. At each eligible time step t, the agent selects a subtask $g_t \sim \pi^h(g; s_t)$ and receives a high-level reward $r^h(s_t, g_t)$ when the subtask terminates after N steps. The high-level reward will be used to optimize the high-level policy via RL. We define $r^h(s_t, g_t)$ as the accumulated low-level rewards from time step t to $t + N$. For intermediate time steps (during which the agent works on a selected subtask), there is no high-level reward.

$$r^h(s_t, g_t) = \begin{cases} \sum_{k=t}^{t+N} r^l_{g_t}(s_k, a_k) & \text{for eligible } t \\ 0 & \text{otherwise} \end{cases}$$

While working on subtask g_t, the agent receives a low-level reward for taking action a_t (i.e., predicting g_t or querying the user):

$$r^l_{g_t}(s_t, a_t) = \begin{cases} 1 & \text{if } a_t = \ell_{g_t} \\ -\beta & \text{if } a_t = \text{AskUser} \\ -1 & \text{otherwise} \end{cases}$$

where ℓ_{g_t} is the ground-truth label for subtask g_t and $\beta \in [0, 1)$ is the penalty for querying the user. The received reward will be used to optimize the low-level policy $\pi^l_{g_t}$ for this subtask via RL.

Essentially, the low-level reward r^l_g alleviates the reward sparsity in the long trajectory of the entire task, and stimulates the agent to predict a correct component with fewer questions. Note that during the course of RL we do not stop the parsing even if one of the predictions is incorrect in order to encourage the agent to predict as more correct components as possible. This also fits the realistic application setting where the agent does not know the ground truth at the component level, and does not receive the external reward signal until it recommends a synthesized program to the user at the end of the interactive parsing process.

Transition. Interactive semantic parsing starts with a state s_0 where $b_i = 0$ (i.e., no subtask is completed) and $d_i = \varnothing$ (i.e., no user answer). As the agent takes actions, it deterministically transits to a new state with updated b_i and d_i.

3.2 Hierarchical Policy Functions

Low-level policy function. The low-level policy $\pi^l_{st_i}(a; s)$ decides whether to ask a question or predict subtask st_i (e.g., selecting a trigger channel name for st_1). When it selects the "AskUser" action, the user will clarify the subtask with a natural language utterance as shown in Table 1. The policy then decides the next action by considering both the recipe description and the user response.

To understand a recipe description, we choose one of the state-of-the-art models, Latent Attention Model (LAM) (Liu et al. 2016). The main idea behind LAM is to first understand the latent sentence structure and then pay attention to words that are critical for a subtask. For example, given a description with a pattern "X to Y," LAM adopts a latent attention

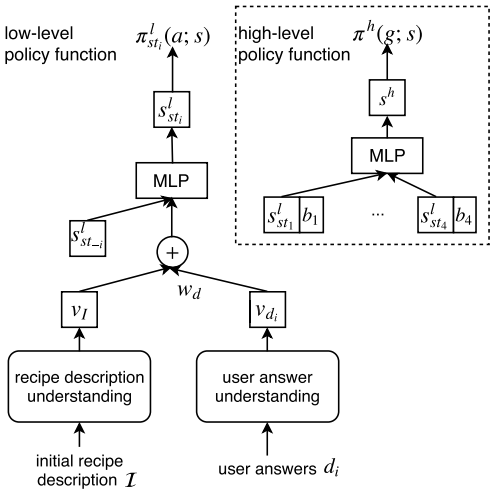


Figure 2: High- and low-level policy designs.

mechanism to first locate the keyword "to," and then focus more on "X" when predicting the trigger channel/function and on "Y" when predicting the action channel/function. This reveals that different subtasks need different policies, so that they can focus on different parts of a recipe description. Such design also allows each subtask to have a different and reduced action space. Hence, we define 4 low-level policy functions with the same model structure yet different parameters for the 4 subtasks, respectively.

To deal with user responses, one straightforward adaptation from LAM is to simply concatenate them with the initial recipe description as the input and extend the prediction space with an extra "AskUser" action. However, we observe that user answers are fundamentally different from a recipe description, e.g., user answers typically contain information related to the queried subtask (rather than all subtasks). Therefore, we propose to model these two parts separately as shown in Figure 2. In particular, we employ LAM to instantiate the "*recipe description understanding*" module to capture the meaning of the initial recipe description (i.e., v_I), and utilize a bidirectional GRU-RNN with the attention mechanism (Zhou et al. 2016) to instantiate the "*user answer understanding*" module.[2] Details can be found in the Appendix A. Since each subtask maintains a separate policy function, the learned v_I can be different for different subtasks. The agent can ask the user to clarify a subtask for multiple times,[3] and the newly received answers will be concatenated with the old ones to update d_i.

We combine the representations of recipe description (v_I) and user answers (v_{d_i}) as the semantic representation related to subtask st_i:

$$v_i = (1 - w_d)v_I + w_d v_{d_i}, \tag{1}$$

where $w_d \in [0, 1]$ is a weight controlling information from the initial recipe description versus from user answers. The semantic vector v_i, concatenated with the information of other subtasks, defines the *low-level state vector* $s^l_{st_i}$ of subtask st_i via a multi-layer perceptron model (the "MLP" module in Figure 2):

$$s^l_{st_i} = \tanh(W_{c_i}[s^l_{st_1}; ...; s^l_{st_{i-1}}; v_i; s^l_{st_{i+1}}; ...; s^l_{st_4}]).^4 \tag{2}$$

Essentially, $s^l_{st_i}$ summarizes the state information for completing subtask st_i, including the initial recipe description, user answers for st_i, and the current low-level state vectors of other subtasks (such that the completion status of other subtasks can affect the current one, as shown in Table 1). Finally, the low-level policy function $\pi^l_{st_i}(a; s)$=softmax$(W^l_{st_i} s^l_{st_i})$, takes the state vector as the input, and outputs a probability distribution over the action space of subtask st_i.

High-level policy function. The high-level policy $\pi^h(g; s)$ receives a state s and decides the next subtask g. The *high-level state vector* s^h is learned to encode the state of overall parsing task through a multi-layer perceptron model (i.e., the "MLP" module in Figure 2) using the 4 low-level state vectors $s^l_{st_i}$'s, as well as the subtasks' boolean indicators b_i's, as inputs:

$$s^h = \tanh(W_c[s^l_{st_1}; b_1; s^l_{st_2}; b_2; s^l_{st_3}; b_3; s^l_{st_4}; b_4]),$$
$$\pi^h(g; s) = \text{softmax}(W^h s^h). \tag{3}$$

Optimization. The high-level policy π^h is trained to maximize the expectation of the discounted cumulative rewards for selecting subtask g_t in state s_t:

$$\max_{\pi^h} J(\theta) = \max_{\pi^h} E_{\pi^h}[r^h(s_t, g_t) \tag{4}$$
$$+ \gamma r^h(s_{t+N_1}, g_{t+N_1}) + \gamma^2 r^h(s_{t+N_1+N_2}, g_{t+N_1+N_2})$$
$$+ ... + \gamma^\infty r^h(s_{t+\sum_{n=1}^{\infty} N_n}, g_{t+\sum_{n=1}^{\infty} N_n})|s_t, g_t, \pi^h(\theta)],$$

where θ stands for parameters in π^h, $N_n (n = 1, 2, ..., \infty)$ is the number of time steps that the agent spent on the previous subtask, and $\gamma \in [0, 1]$ is the discount factor. Similarly, we train the low-level policy $\pi^l_{g_t}$ for the selected subtask g_t to maximize its expected cumulative discounted low-level reward:

$$\max_{\pi^l_{g_t}} J_{g_t}(\phi_{g_t}) = \max_{\pi^l_{g_t}} E_{\pi^l_{g_t}}[\tag{5}$$
$$\sum_{k \geq 0} \gamma^k r^l_{g_t}(s_{t+k}, a_{t+k})|s_t, a_t, g_t, \pi^l_{g_t}(\phi_{g_t})],$$

where ϕ_{g_t} denotes the parameters in $\pi^l_{g_t}$, and γ is the same discount factor.

All policies are stochastic in that the next subtask or action is sampled according to the probability distribution which allows exploration in RL, and that the policies can be optimized using policy gradient methods. In our experiments we used the REINFORCE algorithm (Williams 1992). Details are outlined in Algorithm 1 of the Appendix.

4 Experiments

We experiment with our proposed HRL agent under both simulation and human evaluation.

[2] We also verified this separate design is much better than the straightforward adaptation from LAM mentioned earlier during model development.

[3] Using the same predefined question (see User Simulation).

[4] Bias terms are omitted for clarity.

Test Data	CI	VI		Total
		VI-1/2	VI-3/4	
Size	727	1,271	1,872	3,870
(%)	(18.79)	(32.84)	(48.37)	(100)

Table 2: Statistics of the test subsets.

4.1 Dataset

We utilize the 291,285 <recipe, description> pairs collected by Ur et al. (2016) for training and the 3,870 pairs from Quirk, Mooney, and Galley (2015) for testing.[5] 20% of the training data are randomly sampled as a validation set. All recipes were created by real users on IFTTT.com. In total, the datasets involve 251 trigger channels, 876 trigger functions, 218 action channels and 458 action functions. For each description in the test set, Quirk, Mooney, and Galley (2015) collected five recipe annotations from Amazon Mechanical Turkers. For each subtask, if at least three annotators make the same annotation as the ground truth, we consider this recipe description as *clear* for this subtask; otherwise, it is labeled as *vague*. In this way, we split the entire test set into three subsets as shown in Table 2: (1) *CI*: 727 recipes whose descriptions are clear for all 4 subtasks; (2) *VI-1/2*: 1,271 recipes containing 1 or 2 vague subtasks; (3) *VI-3/4*: 1,872 recipes containing 3 or 4 vague subtasks.

4.2 Methods Comparison

- **LAM**: The Latent Attention Model (Liu et al. 2016),[6] one of the state-of-the-art models for synthesizing If-Then recipes. We do not consider the model ensemble in (Liu et al. 2016), as it can be applied to all other methods as well. Our reproduced LAM obtains a performance close to the reported one without ensemble.
- **LAM-rule Agent**: A rule-based agent built on LAM, similar to (Chaurasia and Mooney 2017). Specifically, the agent makes a prediction on a subtask with a certain probability. If the probability is lower than a threshold,[7] the user is asked a question. The user answer is concatenated with the initial recipe description for making a new prediction. This procedure repeats until the prediction probability is greater than the threshold or the agent has run for Max_Local_Turn turns on the subtask.
- **LAM-sup Agent**: An agent based on LAM, but with the user answer understanding module in Figure 2 and an extra "AskUser" action for each subtask. It is trained via a supervised learning strategy and thus is named LAM-sup. We collected the training data for each subtask st_i based on the LAM-rule agent: If LAM-rule completes the subtask without interactions with humans, we add a tuple $<\mathcal{I}, \varnothing, \ell_{st_i}>$ to the training set, where $\mathcal{I}$ is the recipe description and ℓ_{st_i} is the ground-truth label for subtask st_i; otherwise, we add two tuples $<\mathcal{I}, \varnothing,$ "AskUser"$>$ and $<\mathcal{I}, d_i, \ell_{st_i}>$, where d_i is the received user answer. We train the agent by minimizing the cross entropy loss. During testing, for each recipe description, the agent starts with no user answer; for each subtask, if it predicts the "AskUser" label, the received user answer will be concatenated with previous ones to make a new prediction until a non-AskUser label is selected.
- **HRL Agent**: Our agent with a two-level hierarchical policy.
- **HRL-fixedOrder Agent**: A variant of our HRL agent with a fixed subtask order of "st_1-st_2-st_3-st_4" and no high-level policy learning.

Evaluation metrics. We compare each method on three metrics: (1) *C+F Accuracy*: A recipe is considered parsed correctly only when all its 4 components (i.e., Channel+Function) are accurately predicted, as adopted in (Quirk, Mooney, and Galley 2015; Liu et al. 2016). (2) *Overall Accuracy*: We measure the average correctness of predicting 4 components of a recipe, e.g., the overall accuracy for predicting 3 components correctly and 1 incorrectly is 0.75. (3) *#Asks*: The averaged number of questions for completing an entire task. Generally, the *C+F Accuracy* is more challenging as it requires no mistake on any subtask. On the other hand, *#Asks* can reveal if an agent asks redundant questions. In our experiments, we consider *C+F Accuracy* and *#Asks* as two primary metrics.

Implementation details. The word vector dimension is set at 50, the weight factor w_d is 0.5, and the discount factor γ is 0.99. The max turn Max_Local_Turn is set at 5 and Max_Global_Turn at 4, which allows four subtasks at most. β is a trade-off between parsing accuracy and the number of questions: A larger β trains an agent to ask fewer questions but with less accuracy, while a lower β leads to more questions and likely more accurate parses. With the validation set, we experimented with β=$\{0.3, 0.4, 0.5\}$, and observed that when $\beta = 0.3$, the number of questions raised by the HRL-based agents is still reasonable compared with LAM-rule/sup, and its parsing accuracy is much higher. More details are shown in Appendix C.

4.3 User Simulation

In our work, for each subtask, agent questions are predefined based on templates,[8] and a user answer is a natural language description about the queried subtask. Given that it is too costly to involve humans in the training process, we introduce a user simulator to provide answers to agent questions.

For a trigger/action function, we adopt several strategies to simulate user answers, including revising its official description on IFTTT.com and replacing words and phrases in the function description with their paraphrases according to the PPDB paraphrase database (Pavlick et al. 2015).

[5] Quirk, Mooney, and Galley (2015) only released the urls of recipes in their test set, among which the unavailable ones have been removed from our test set. Each recipe is associated with a unique ID. We ensure no overlapping recipes between training and testing set by examining their IDs.

[6] Unlike (Liu et al. 2016), which consolidates channel and function names (e.g., "`Twitter.New liked tweet by you`") and builds 2 classifiers for trigger and action respectively, we develop 4 classifiers so that an agent can inquire channel or function separately.

[7] We set it at 0.85 based on validation set.

[8] Automatically generating recipe-specific questions is non-trivial, which we leave for future work.

Model	Simulation Eval									Human Eval	
	All			CI		VI-1/2		VI-3/4		VI-3/4	
	C+F Acc	Overall Acc	#Asks	C+F Acc	#Asks	C+F Acc	#Asks	C+F Acc	#Asks	C+F Acc	#Asks
LAM	0.374	0.640	0	0.801	0	0.436	0	0.166	0	0.206	0
LAM-rule	0.761	0.926	3.891	0.897	1.433	0.743	2.826	0.721	5.568	0.518	2.781
LAM-sup	0.809	0.940	**2.028**	0.894	**0.684**	0.803	**1.482**	0.780	2.921	0.433	2.614
HRL-fixedOrder	0.881	0.966	2.272	**0.950**	1.522	0.855	1.958	0.871	2.777	0.581	2.306*
HRL	**0.894***	**0.968**	2.069*	0.949	1.226*	**0.888***	1.748*	**0.878***	**2.615***	**0.634***	**2.221***

Table 3: Model evaluation on the test set. For Simulation Eval, each number is averaged over 10 runs. For Human Eval, the LAM result is calculated on the sampled 496 recipes. * denotes significant difference in mean between HRL-fixedOrder vs. HRL in Simulation Eval and between HRL-based agents vs. {LAM-rule, LAM-sup} agents in Human Eval ($p < 0.05$).

In addition, we extract user descriptions of a function from our training set, based on six manually defined templates. For example, for a recipe description following pattern "If X then Y," X will be considered as an answer to questions about the ground-truth trigger function and Y as an answer to those about the ground-truth action function. More details regarding the strategies are in Appendix and source code will also be released. For each function, we collected around 20 simulated user answers on average. In our simulations, for each question we randomly select one from this set of possible answers as a response.

For trigger/action channels, when an agent asks a question, the user simulator will simply provide the channel name (e.g., `GMail`), since it is straightforward and natural for real users as well.

4.4 Simulation Evaluation

Table 3 shows results on the test set in the simulation environment, where our user simulator provides an answer when requested. By enabling the user to clarify, all agents obtain much better accuracy compared with the original non-interactive LAM model. In particular, HRL-based agents outperform others by **7%** $\sim$ **13%** on C+F accuracy and **2%** $\sim$ **4%** in terms of Overall accuracy. For vague recipes in VI-1/2 and VI-3/4 subsets, which make up more than 80% of the entire test set, the advantage of HRL-based agents is more prominent. For example, on VI-3/4, HRL-based agents obtain 9% $\sim$ 15% better C+F accuracy than LAM-rule/sup, yet with fewer questions, indicating that they are much more able to handle ambiguous recipe descriptions.

Compared with HRL-based agents, the LAM-rule agent usually asks the most questions, partly because it relies on heuristic thresholding to make decisions. On VI-3/4, it asks twice the number of questions but parses with 15% less accuracy than HRL-based agents. On the other hand, the LAM-sup agent always asks the least questions, especially when the recipe description is relatively clear (i.e., CI and VI-1/2). However, it may simply miss many necessary questions, leading to at least 5% accuracy loss.

Finally, we evaluate the high-level policy by comparing HRL with HRL-fixedOrder. The significance test shows that HRL requires fewer questions to obtain a similar or better accuracy. Interestingly, we observe that, under the interactive environment, HRL tends to predict the function before the channel, which is different from the inter-task (in)dependence assumptions in previous work. This is mainly because users' descriptions of a function can be more specific and may contain information about its channel. The HRL agent is thus trained to utilize this intuition for asking fewer questions.

4.5 Human Evaluation

We further conduct human evaluation to test the four interactive agents on the most challenging VI-3/4 subset. Two students familiar with IFTTT were instructed to participate in the test. For each session, a recipe from VI-3/4 and one agent from {LAM-rule, LAM-sup, HRL-fixedOrder, HRL} were randomly picked. The participants were presented the description and the ground-truth program components of the recipe, and were instructed to answer clarification questions prompted by the agent with a natural language sentence. To help the participants better understand the recipe, we also showed them the official explanation of each program component. However, we always encouraged them to describe a component in their own words when being asked. For a better user experience and an easier comparison, we limited each agent to ask at most one question for each subtask. In total, we collected 496 conversations between real humans and the four agents. Examples are shown in Appendix E.

We compare each agent primarily on *C+F Accuracy* and *#Asks*. As shown in Table 3, all agents perform much better than the non-interactive LAM model. Particularly, the HRL agent outperforms the LAM model by >40% accuracy, with an average of $\sim$2.2 questions on VI-3/4 (which is a reasonable number of questions as each task contains at least 3 vague subtasks). We also observe that the two HRL-based agents obtain 6% $\sim$ 20% better parsing accuracy with even fewer questions than the LAM-rule/sup agents. Moreover, in comparison with the HRL-fixedOrder agent, the HRL agent can synthesize programs with a much better accuracy but fewer questions, showing the benefit of optimizing subtask order at the high level. However, there is still large space to improve compared with simulation results in Table 3, mainly because agents are trained with the user simulator while the language used by real users for answers can be very different (e.g., having the Out-Of-Vocabulary issue, misspellings,

less or non- relevant information). How to simulate user responses as close as possible to real ones for training is a non-trivial task, which we leave for future work.

4.6 Discussion

Here we further discuss our framework and future work on the following aspects:

Error analysis. Due to the discrepancy between real users and simulated users, several major factors affect the performance of the HRL agent, including user typos (e.g., "emial" for "email") and unseen expressions (e.g., "i tweeted something" to describe the function `New tweet by you`). To improve the robustness of the HRL agent, possible solutions as future work can be modeling user noises in simulation (Li et al. 2016b), or crowdsourcing more diverse component descriptions as user answers for training.

Training with real users in the loop. Theoretically, our agents can be trained with real users. However, it is too costly to be practical because the agent can require many interactions during the training phase. An alternative way is to train the agent in simulation and fine-tune it with real users, or to build a *world model* that mimics real user behaviors during the human-in-the-loop training (Peng et al. 2018). Both approaches need significant efforts to be carefully designed, which we leave as future work.

Generalizability to other semantic parsing tasks. The proposed HRL framework can be easily generalized to resolve ambiguities in other semantic parsing tasks where subtasks can be pre-defined. For example, in the knowledge-graph-based question answering task (Berant et al. 2013; Yih et al. 2015), the subtasks include identifying entities, predicting relations, and associating constraints. To train the HRL agent, one can build a user simulator by paraphrasing the ground-truth entities or relations, similar to Section 4.3. HRL can be very promising for these tasks, as it enables temporal abstractions over the state and action space (leading to a smaller search space) and can model the dependencies between subtasks, as shown in Table 1. We will explore these applications in the future.

5 Related Work

In addition to the aforementioned work on If-Then program synthesis, Dong, Quirk, and Lapata (2018) investigated how to measure a semantic parser's confidence in its predictions, but did not further resolve uncertainties. Others include:

Interactive Systems for Resolving Ambiguities. Resolving ambiguities via interactions with humans has been explored in Natural Language Understanding in dialog systems (Thomason et al. 2015; Dhingra et al. 2017), Question Answering (Guo et al. 2016; Li et al. 2017), CCG parsing (He et al. 2016) and parsing for SQL and web APIs (Li and Jagadish 2014; Gur et al. 2018; Su et al. 2018). Guo et al. (2016) built an agent to ask relevant questions until it has enough information to correctly answer user's question, but expected the user to respond with an oracle value. He et al. (2016) investigated generating multi-choice questions for humans to resolve uncertainties in parsing sentences. They determined the necessity of a question by a heuristic threshold. In contrast, we allow users to respond with natural language utterances, and our HRL-based agents can learn when to ask through the reward mechanism. Recently, (Li et al. 2016a; Azaria, Krishnamurthy, and Mitchell 2016; Iyer et al. 2017) explored human feedback on the final results as training supervision. Different from theirs, we include humans during the parsing process for them to provide necessary information in natural language, and define rewards as the only feedback.

Hierarchical Reinforcement Learning (HRL). To solve a complex task, HRL decomposes the task into several easier subtasks and solve them sequentially via MDPs (Parr and Russell 1998; Sutton, Precup, and Singh 1999; Dietterich 2000). Recently, HRL-based approaches are applied to tasks like game playing (Kulkarni et al. 2016; Tessler et al. 2017), travel planning (Peng et al. 2017), and visual question answering and captioning (Wang et al. 2017; Gordon et al. 2017; Zhang, Zhao, and Yu 2018). Inspired by these work, given that our semantic parsing task can be naturally decomposed into 4 subtasks, we learn a two-level policy where a high-level policy decides the subtask order while a low-level policy accomplishes each subtask by asking humans clarifying questions if necessary.

6 Conclusion

In this paper we explored using HRL for *interactive semantic parsing*, where an agent asks clarification questions when the initially given natural language description is ambiguous and accomplishes subtasks in an optimized order. On the If-Then recipe synthesis task, in both simulation and human evaluation settings, we have shown that our HRL agent can substantially outperform various interactive baselines in that it produces more accurate recipes but asks the user fewer questions in general. As future work, we will generalize our HRL framework to other semantic parsing tasks such as knowledge based question answering, explore better training strategies such as modeling real user noises in simulation, as well as further reduce the user interaction turns.

Acknowledgments

We would like to thank Chris Brockett, Michel Galley and Yu Su for their insightful comments on the work. This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, Fujitsu gift grant, and Ohio Supercomputer Center (Center 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Azaria, A.; Krishnamurthy, J.; and Mitchell, T. M. 2016. Instructable intelligent personal agent. In *AAAI*, 2681–2689.

Beltagy, I., and Quirk, C. 2016. Improved semantic parsers for if-then statements. In *ACL*, volume 1, 726–736.

Berant, J.; Chou, A.; Frostig, R.; and Liang, P. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 1533–1544.

Campagna, G.; Ramesh, R.; Xu, S.; Fischer, M.; and Lam, M. S. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *WWW*, 341–350.

Center, O. S. 1987. Ohio supercomputer center. `http://osc.edu/ark:/19495/f5s1ph73`.

Chaurasia, S., and Mooney, R. J. 2017. Dialog for language to code. In *IJCNLP*, volume 2, 175–180.

Dhingra, B.; Li, L.; Li, X.; Gao, J.; Chen, Y.-N.; Ahmed, F.; and Deng, L. 2017. Towards end-to-end reinforcement learning of dialogue agents for information access. In *ACL*.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Dong, L., and Lapata, M. 2016. Language to logical form with neural attention. In *ACL*, volume 1, 33–43.

Dong, L.; Quirk, C.; and Lapata, M. 2018. Confidence modeling for neural semantic parsing. In *ACL*.

Gao, J.; Galley, M.; and Li, L. 2018. Neural approaches to conversational ai. *arXiv preprint arXiv:1809.08267*.

Gordon, D.; Kembhavi, A.; Rastegari, M.; Redmon, J.; Fox, D.; and Farhadi, A. 2017. Iqa: Visual question answering in interactive environments. *arXiv preprint arXiv:1712.03316*.

Guo, X.; Klinger, T.; Rosenbaum, C.; Bigus, J. P.; Campbell, M.; Kawas, B.; Talamadupula, K.; Tesauro, G.; and Singh, S. 2016. Learning to query, reason, and answer questions on ambiguous texts.

Gur, I.; Yavuz, S.; Su, Y.; and Yan, X. 2018. Dialsql: Dialogue based structured query generation. In *ACL*, 1339–1349.

He, L.; Michael, J.; Lewis, M.; and Zettlemoyer, L. 2016. Human-in-the-loop parsing. In *EMNLP*, 2337–2342.

Iyer, S.; Konstas, I.; Cheung, A.; Krishnamurthy, J.; and Zettlemoyer, L. 2017. Learning a neural semantic parser from user feedback. In *ACL*, volume 1, 963–973.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*.

Li, F., and Jagadish, H. 2014. Constructing an interactive natural language interface for relational databases. *VLDB*.

Li, J.; Miller, A. H.; Chopra, S.; Ranzato, M.; and Weston, J. 2016a. Dialogue learning with human-in-the-loop. *arXiv preprint arXiv:1611.09823*.

Li, X.; Lipton, Z. C.; Dhingra, B.; Li, L.; Gao, J.; and Chen, Y.-N. 2016b. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*.

Li, H.; Min, M. R.; Ge, Y.; and Kadav, A. 2017. A context-aware attention network for interactive question answering. In *SIGKDD*, 927–935. ACM.

Liu, C.; Chen, X.; Shin, E. C.; Chen, M.; and Song, D. 2016. Latent attention for if-then program synthesis. In *NIPS*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Parr, R., and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *NIPS*, 1043–1049.

Pavlick, E.; Rastogi, P.; Ganitkevitch, J.; Van Durme, B.; and Callison-Burch, C. 2015. Ppdb 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *ACL-IJCNLP*, 425–430.

Peng, B.; Li, X.; Li, L.; Gao, J.; Celikyilmaz, A.; Lee, S.; and Wong, K.-F. 2017. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *EMNLP*, 2221–2230.

Peng, B.; Li, X.; Gao, J.; Liu, J.; and Wong, K.-F. 2018. Deep dyna-q: Integrating planning for task-completion dialogue policy learning. In *ACL*, volume 1, 2182–2192.

Quirk, C.; Mooney, R. J.; and Galley, M. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In *ACL*, 878–888.

Rabinovich, M.; Stern, M.; and Klein, D. 2017. Abstract syntax networks for code generation and semantic parsing. In *ACL*, volume 1, 1139–1149.

Su, Y.; Awadallah, A. H.; Khabsa, M.; Pantel, P.; Gamon, M.; and Encarnacion, M. 2017. Building natural language interfaces to web apis. In *CIKM*, 177–186. ACM.

Su, Y.; Awadallah, A. H.; Wang, M.; and White, R. W. 2018. Natural language interfaces with fine-grained user interaction: A case study on web apis. In *SIGIR*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2):181–211.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, volume 3, 6.

Thomason, J.; Zhang, S.; Mooney, R. J.; and Stone, P. 2015. Learning to interpret natural language commands through human-robot dialog. In *IJCAI*, 1923–1929.

Ur, B.; Pak Yong Ho, M.; Brawner, S.; Lee, J.; Mennicken, S.; Picard, N.; Schulze, D.; and Littman, M. L. 2016. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *CHI*, 3227–3231. ACM.

Wang, X.; Chen, W.; Wu, J.; Wang, Y.-F.; and Wang, W. Y. 2017. Video captioning via hierarchical reinforcement learning. *arXiv preprint arXiv:1711.11135*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*. Springer. 5–32.

Yih, S. W.-t.; Chang, M.-W.; He, X.; and Gao, J. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base.

Yin, P., and Neubig, G. 2017. A syntactic neural model for general-purpose code generation. *arXiv:1704.01696*.

Zhang, J.; Zhao, T.; and Yu, Z. 2018. Multimodal hierarchical reinforcement learning policy for task-oriented visual dialog. *arXiv preprint arXiv:1805.03257*.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR* abs/1709.00103.

Zhou, P.; Shi, W.; Tian, J.; Qi, Z.; Li, B.; Hao, H.; and Xu, B. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *ACL*, 207–212.

A User Answer Understanding Module

We define the user answer understanding module in Figure 2 based on a bidirectional Recurrent Neural Network with attention. Specifically, for the j-th word in the user answer d_i, we concatenate its forward and backward hidden states (i.e., $h_{d_{i,j}} = [\overrightarrow{h}_{d_{i,j}}; \overleftarrow{h}_{d_{i,j}}]$) as its semantic representation. Attention weights w_{att_j} over all words are computed with a trainable context vector c, i.e., $w_{att_j} = \text{softmax}(c^T h_{d_{i,j}})$, which will help the agent identify important words in d_i. User answer d_i is then represented as $v_{d_i} = \sum_j w_{\text{att}_j} h_{d_{i,j}}$.

B Training Algorithm for HRL

In our work, policy functions are all updated via Stochastic Gradient Ascent. Given the objective Eq (4), we deduce the updates for the high-level policy function below:

$$\nabla_\theta J(\theta) = E_{\pi^h}[\nabla_\theta \log \pi^h(g; s) u_t], \tag{6}$$

where $u_t = \sum_{\kappa=0}^{\infty} \gamma^\kappa r^h(s_{t+\sum_{n=1}^{\kappa} N_n}, g_{t+\sum_{n=1}^{\kappa} N_n})$ is the summation item within the two brackets in Eq (4).

Similarly, at low level, for the ongoing subtask $g_t \in \mathcal{G} = \{st_1, st_2, st_3, st_4\}$ at time step t, the updates of its policy function $\pi^l_{g_t}$ are given by:

$$\nabla_{\phi_{g_t}} J_{g_t}(\phi_{g_t}) = E_{\pi^l_{g_t}}[\nabla_{\phi_{g_t}} \log \pi^l_{g_t}(a; s) u_{g_t,t}], \tag{7}$$

where $u_{g_t,t} = \sum_{k \geq 0} \gamma^k r^l_{g_{t+k}}(s_{t+k}, a_{t+k})$.

Algorithm 1 details the entire training procedure. Line 1-2 initialize each policy network. Particularly, each low-level policy network is pre-trained via supervised learning (Section 4). We train the agent on M episodes (i.e., recipes). At the beginning of each episode, the state vector $s^l_{st_i}$ is initialized by calculating Eq (1-2) with $s^l_{st_{-i}} = \overrightarrow{0}$ (Line 7). Gradients of parameters θ and ϕ_{st_i} are accumulated during every episode. To improve efficiency, we perform gradient update for every 64 episodes (Line 31-34).

C Implementation Details

To accelerate the training of HRL-based agents, we adopt a similar skill as (Gordon et al. 2017; Wang et al. 2017) to initialize their low-level policy functions with the well-trained LAM-sup model. Particularly for HRL, we apply an "action mask" at the high level, so that the completed subtask will not be repeatedly chosen as the next subtask. However, the agent does not rely on this mask after being fully trained. The validation check is performed every 2K iterations on a randomly sampled 1K validation recipes. The best parameter setting is the one that obtains the most high-level reward in validation.

D User Simulator

For a trigger/action function, we design the following strategies to simulate user answers:

1. Revised function description: Each function has an official description on IFTTT.com, e.g., for function `Create a link note`, its description is "*This action will create a new link note from a source URL in the notebook you specify*." We revised the function description to make a simulated user answer, e.g., "*Create a new link note from a source URL in the notebook I specify*."
2. Paraphrases of the revised function description: Specifically, we replaced words and phrases with their paraphrases, which were collected and marked as semantically "Equivalence" by the PPDB paraphrase database (Pavlick et al. 2015).
3. Extractions from a large collection of real-user recipe descriptions in our training set (Ur et al. 2016): The extractions are based on six manually defined templates and will embrace language variation in simulated user answers. For example, for a recipe description "If X then Y," X will be considered as an answer to questions about the ground-truth trigger function and Y as an answer to those about the ground-truth action function.

E User Evaluation Examples

We present three user evaluation examples in Table 4.

Example 1
User: "pinterest post to wordpress" **Ground-truth recipe**: [tc: `Feed`, tf: `New feed item`, ac: `WordPress`, af: `Create a photo post`] **HRL agent**: "Which event triggers the action?" **User**: "when there's a new pinterest post' **HRL agent**: "Which event results from the trigger?" **User**: "create a post with photo" **Agent Prediction**: [tc: `Feed`, tf: `New feed item`, ac: `WordPress`, af: `Create a photo post`]
Example 2
User: "foto do instagram" **Ground-truth recipe**: [tc: `Instagram`, tf: `Any new photo by you`, ac: `Email`, af: `Send me an email`] **HRL agent**: "Which event triggers the action?" **User**: "I share new ins photo" **HRL agent**: "Which channel should act per your request?" **User**: "email" **Agent Prediction**: [tc: `Instagram`, tf: `Any new photo by you`, ac: `Email`, af: `Send me an email`]
Example 3
User: "send fiverr orders to wunderlist" **Ground-truth recipe**: [tc: `Fiverr`, tf: `New order received`, ac: `Gmail`, af: `Send an email`] **HRL agent**: "Which event triggers the action?" **User**: "a new order is recievied" **Agent Prediction**: [tc: `Fiverr`, tf: `New email in inbox from search`, ac: `Gmail`, af: `Send an email`]

Table 4: Two examples (Example 1-2) from user evaluation where the HRL agent correctly interpreted the user instructions and two (Example 3) where it did not. The agent failed when there is a typo in the word "recievied".

Algorithm 1 Learning algorithm for HRL

Initialize parameters θ of the high-level policy network randomly.
Initialize parameters ϕ_{st_i} $(i = 1, 2, 3, 4)$ of each low-level policy network with supervised pre-training.
Initialize gradients: $d\theta \leftarrow 0$, $d\phi_{st_i} \leftarrow 0$.
for $\#episode = 1$ to M **do**
    Reset the user simulator and get a recipe description $\mathcal{I}$.
    Initialize $b_i \leftarrow 0, d_i \leftarrow \emptyset, \forall i = 1, 2, 3, 4$. Observe $s_0 = \{\mathcal{I}, b_i, d_i\}$.
    Calculate $s^l_{st_i}, \forall i = 1, 2, 3, 4$, according to Eq (1-2), with $s^l_{st_{-i}} = \overrightarrow{0}$.
    $global\_turn \leftarrow 1$.
    $t \leftarrow 0$.
    while s_t is not terminal **and** $global\_turn \leq Max\_Global\_Turn$ **do**
        Sample a subtask $g_t \sim \pi^h(g; s_t)$ according to Eq (3). We denote g_t as st_{i_t}, i.e., the i_t-th subtask in the subtask set $\mathcal{G} = \{st_1, st_2, st_3, st_4\}$.
        $t_{start} \leftarrow t$.
        $local\_turn \leftarrow 1$, $a_t \leftarrow \varnothing$.
        while ($a_t == \varnothing$ **or** $a_t ==$ "AskUser") **and** $local\_turn \leq Max\_Local\_Turn$ **do**
            Sample a primitive action $a_t \sim \pi^l_{st_{i_t}}(a; s_t)$.
            Execute and receive a low-level reward $r^l_{st_{i_t}}(s_t, a_t)$.
            $d_{i_t} \leftarrow d_{i_t} \cup$ retrieved simulated user answer.
            Observe new state s_{t+1} with new d_{i_t}.
            Update $s^l_{st_{i_t}}$ according to Eq (1-2).
            $g_{t+1} \leftarrow g_t$.
            $t \leftarrow t + 1$.
            $local\_turn \leftarrow local\_turn + 1$.
        end while
        $d\phi_{st_{i_t}} \leftarrow d\phi_{st_{i_t}} +$ accumulated gradient according to Eq (7).
        Receive a high-level reward $r^h(s_{t_{start}}, g_{t_{start}})$.
        $b_{i_t} \leftarrow 1$.
        $t \leftarrow t + 1$. Observe state s_t.
        $global\_turn \leftarrow global\_turn + 1$.
    end while
    $d\theta \leftarrow d\theta +$ accumulated gradient according to Eq (6).
    if $\#episode \ \% \ 64 \ == 0$ **then**
        Perform update of θ and ϕ_{st_i} $(i = 1, 2, 3, 4)$ by gradient ascent using $d\theta$ and $d\phi_{st_i}$.
        $d\theta \leftarrow 0$, $d\phi_{st_i} \leftarrow 0$.
    end if
end for

RIKER: Mining Rich Keyword Representations for Interpretable Product Question Answering

Jie Zhao
The Ohio State University
zhao.1359@osu.edu

Ziyu Guan
Xidian University
zyguan@xidian.edu.cn

Huan Sun
The Ohio State University
sun.397@osu.com

ABSTRACT

This work studies product question answering (PQA) which aims to answer product-related questions based on customer reviews. Most recent PQA approaches adopt end2end semantic matching methodologies, which map questions and answers to a latent vector space to measure their relevance. Such methods often achieve superior performance but it tends to be difficult to interpret why. On the other hand, simple keyword-based search methods exhibit natural interpretability through matched keywords, but often suffer from the lexical gap problem. In this work, we develop a new PQA framework (named RIKER) that enjoys the benefits of both interpretability and effectiveness. RIKER mines rich keyword representations of a question with two major components, *internal word re-weighting* and *external word association*, which predict the importance of each question word and associate the question with outside relevant keywords respectively, and can be jointly trained under weak supervision with large-scale QA pairs. The keyword representations from RIKER can be directly used as input to a keyword-based search module, enabling the whole process to be effective while preserving good interpretability. We conduct extensive experiments using Amazon QA and review datasets from 5 different departments, and our results show that RIKER substantially outperforms previous state-of-the-art methods in both synthetic settings and real user evaluations. In addition, we compare keyword representations from RIKER and those from attention mechanisms popularly used for deep neural networks through case studies, showing that the former are more effective and interpretable.

KEYWORDS

Product QA, Interpretable Search, Question Representation

ACM Reference Format:
Jie Zhao, Ziyu Guan, and Huan Sun. 2019. RIKER: Mining Rich Keyword Representations for Interpretable Product Question Answering. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3292500.3330985

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330985

1 INTRODUCTION

Question answering (QA) on e-commerce websites allows customers to acquire useful information for making purchase decisions. Currently, answering product-related questions still largely relies on costly human efforts. It can take several days to get an answer from sales representatives or other experienced customers. In this work, we study the problem of automating product question answering (PQA) [7]: Given a question regarding a product, we aim to return relevant sentences extracted from corresponding customer reviews to provide answer information. One important characteristic of PQA in comparison with open domain answer selection (e.g., [45, 47, 51]) is its high demand in system interpretability, partly because customers may have increasing concerns about hidden advertising agendas and leaked personal information and can distrust a PQA system [18] if puzzled by how they get the results, and also partly because latest policies and regulations are also urging companies to provide interpretations for their algorithms that directly affect users [13, 17].

To the best of our knowledge, all recent PQA works [30, 43, 48] advocate end2end semantic matching methodologies, which tend to be black-box and directly output a matching score for each <question, review sentence> pair. Typically, questions/answers/reviews are first encoded into low-dimensional vector representations, which are then used to generate the matching scores based on some relevance functions (e.g., dot product). For example, question-review and answer-review encoders and relevance functions are simultaneously learned by optimizing a mixture-of-experts objective [30]. In fact, in open domain as well, almost all latest answer selection works [41, 42, 50] fall into this category, e.g., relying on hidden units within deep neural networks (DNN) to represent the relevance of QA sentences pairs [28]. Despite its popularity, the end2end paradigm has received challenges on its lack of interpretability [14]. Even with many recent efforts [3, 5, 10, 37, 39], it is still very difficult to interpret dense vector representations or how an answer is matched with a question. Meanwhile, different attention mechanisms [31, 41, 50] have been incorporated into DNN models to associate the relevant parts across two sentences. Although to some extent the intermediate attentions can help reveal the soft word alignments between two sentences, the models are still less transparent because of the nonlinear relationship between attention scores and outputs [35].

In this work, we do not follow the end2end paradigm of existing approaches, and instead advocate a hybrid framework marrying the advantages of both DNN structures and classic keyword-based Information Retrieval (IR) techniques [4]. Keyword-based IR techniques such as *tf-idf* ranking functions *naturally exhibit much better interpretability* owing to their transparency and intuitiveness:

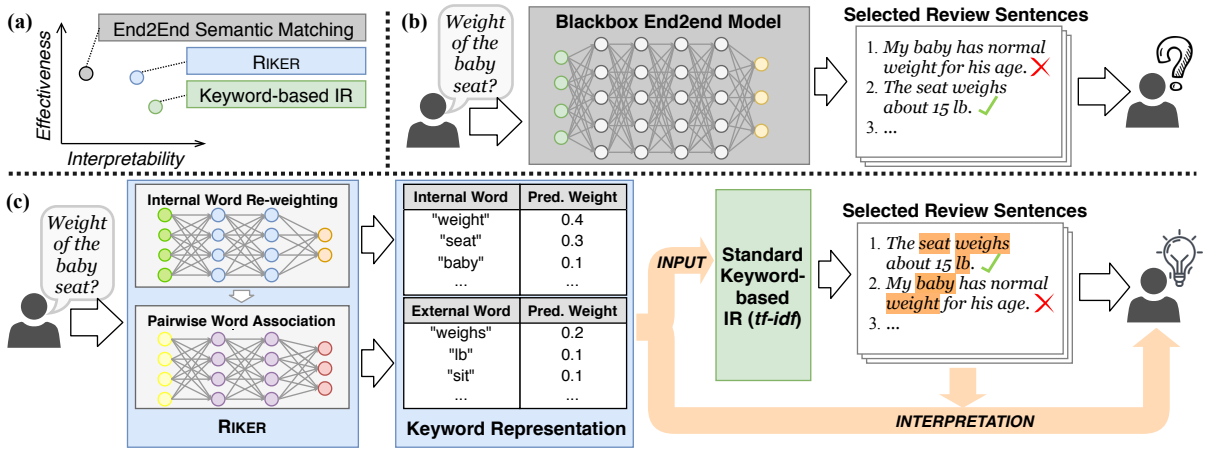


Figure 1: Interpretability and effectiveness of different methodologies. Here we use DNN structures as an example of end2end methods. Our proposed method RIKER mines rich keyword representations for a question, which are used as input to a standard *tf-idf* based IR module. With overlapped words highlighted in orange, such representations also serve as interpretation to the user for the retrieved results and help keep the ranking process transparent.

An answer candidate will be ranked higher if it has more important keywords matched with the question. However, standalone keyword-based IR methods are often not as effective due to the lexical gap problem [38], i.e., they cannot handle mismatched yet relevant words. In addition, they tend to weigh question words only according to their corpus-level statistics like *tf-idf* without considering the specific sentence-level context information. Therefore, to achieve a good balance between interpretability and effectiveness, *we propose a new framework named RIKER, which mines rich keyword representations of a question and uses them to effectively improve the performance of interpretable keyword-based search techniques.*

Figure 1(a) qualitatively shows the trade-off between effectiveness and interpretability for different methodologies, while Figure 1(b) and 1(c) demonstrate how an end2end model and RIKER process the same question *"Weight of the baby seat?"*. An end2end system is often hard for humans to understand or intervene. As exemplified in Figure 1(b), when an irrelevant review sentence is ranked at the top (because the system misinterprets the question intent as "*baby weight*" rather than "*baby seat weight*"), the user may have no choice but keep rephrasing the question to probe the system. In contrast, as Figure 1(c) shows, RIKER can map the question into its keyword representations, which emphasize important internal words (i.e., words that appear in the question) such as *"weight"*, *"seat"* and include relevant external words (i.e., words that are not in the question but exist in the corpus) such as *"weighs"* and *"lb"*. Such easy-to-understand keyword representations can then be used as input to standard *tf-idf* based IR, which keeps the review sentence selection process transparent. Moreover, if unsatisfied with current results, the user can adjust the keywords and their predicted weights (e.g., set the predicted weight of "*baby*" to zero to reduce ambiguity) and anticipate better results.

Specifically, RIKER consists of two neural components: (1) Internal word re-weighting, which predicts the relative importance of words *within* the question, and assigns more weights to those that are more likely to occur in correct answers and less likely in incorrect ones; (2) External word association, which models pairwise word associations between questions and answers, and enables RIKER to infer relevant keywords *outside* the question in the entire vocabulary. For training, we design two novel objective functions respectively for each component. They explicitly encourage assigning higher scores to matched or associated keywords between correct QA pairs and mutually enhance each other, so that RIKER achieves the best overall performance when both objectives are jointly optimized. The mined keyword representations are employed as input to the IR module to alleviate its lexical gap problem and enhance the search performance, and are demonstrated more effective than the state-of-the-art word embedding based query expansion methods [12, 49]. At the same time, they can serve as interpretations to human users and allow RIKER to enjoy the interpretability of the keyword-based search techniques, which we demonstrate both quantitatively and qualitatively through case studies.

In summary, our contributions are as follows:

- To the best of our knowledge, we are the first to advocate interpretability in product QA, which can give customers a better intuition of how their questions are answered and eventually helps earn user trust in the system.
- We design a new PQA framework, RIKER, which jointly identifies important keywords within the question and associates relevant words outside the question based on neural models. Such keyword representations can improve the search performance and meanwhile preserve the interpretability of a keyword-based search module.
- We conduct extensive experiments using 5 PQA datasets from *Amazon.com* as well as a real user study, and show that our framework consistently outperforms existing query expansion and PQA methods. In addition, we also discuss RIKER w.r.t. a widely adopted attention mechanism for DNN models on their interpretability via case studies.

2 OVERVIEW

Traditional keyword-based IR techniques widely used by e-commerce websites have the advantage of being self-explanatory to general

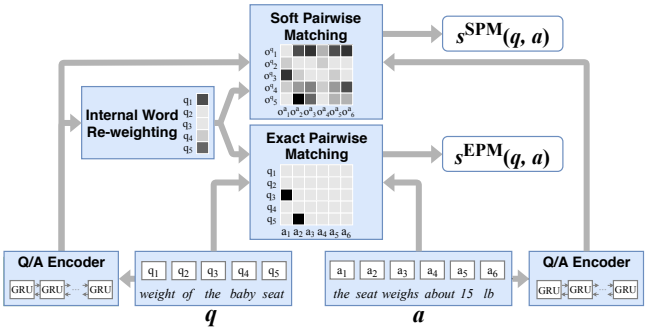


Figure 2: Training structure of Riker.

customers without domain knowledge, and are commonly acknowledged by data scientists as interpretable algorithms among decision trees, rule lists [44], sparse linear models [26], etc. However, naive keyword-based search has drawbacks when applied to PQA: They tend to ignore the relative importance of words given the question context and can not relate different but semantically similar words. To solve this problem, we design Riker that mines better keyword representations for questions. Figure 2 shows the *training* neural architecture of Riker. The *internal word re-weighting* component is learned through exact pairwise matching (Section 3) and predicts higher weights for question words that may exactly appear in the correct answers but rarely in others, whereas the *external word association* component is learned through soft pairwise matching (Section 4) and captures semantic associations between different words. Such keyword representations can preserve the natural interpretability of keyword-based search methods, and also make Riker more effective as experiments will show (Section 5). Additionally, we use case studies to further discuss the interpretability of keyword representations from Riker (Section 6).

3 INTERNAL WORD RE-WEIGHTING

The goal of internal word re-weighting component is to assign different weights to words *within* a given question, according to their relative importance in the context. In this section, we first introduce the details of the re-weighting function, and then describe how it can be trained with an exact pairwise matching objective.

The internal word re-weighting function takes a question q that is a word sequence $[q_1, ..., q_n]$ as input, and outputs a weight for each word, denoted as $f^{\text{int}}(q_i|q)$. As shown in the left part of Figure 2, the question words are mapped to word embeddings and then encoded with a standard bidirectional recurrent neural network (Bi-RNN) with Gated Recurrent Units (GRU) [11] to aggregate sentence-level context information. Specifically, the i-th time step operation of a forward GRU cell parameterized by $\{\vec{W}_r, \vec{b}_r, \vec{W}_u, \vec{b}_u, \vec{W}, \vec{b}\}$is:

$$\begin{aligned} &\vec{r}=\sigma(\vec{W}_r[e(q_i), \vec{h}_{i-1}]+\vec{b}_r) \;;\; \vec{u}=\sigma(\vec{W}_u[e(q_i), \vec{h}_{i-1}]+\vec{b}_u) \\ &\vec{\tilde{h}}_i=\phi(\vec{W}[e(q_i), \vec{r}\odot\vec{h}_{i-1}]+\vec{b}) \;;\; \vec{h}_i=\vec{u}\odot\vec{h}_{i-1}+(1-\vec{u})\odot\vec{\tilde{h}}_i \end{aligned} \quad (1)$$

Here $[\cdot,\cdot]$ represents vector concatenation, $e(q_i)$ is the looked-up word embedding of word q_i, σ and ϕ represent *sigmoid* and *tanh* activation functions respectively, and $\odot$ is element-wise product. A backward GRU cell is defined symmetrically to encode the question from the last word to the first, with a different set of parameters $\{\overleftarrow{W}_r, \overleftarrow{b}_r, \overleftarrow{W}_u, \overleftarrow{b}_u, \overleftarrow{W}, \overleftarrow{b}\}$. The forward and backward GRU hidden states are concatenated at each position to be the output of the Bi-RNN encoder, i.e., $o_i^q = [\vec{h}_i, \overleftarrow{h}_i]$ for q_i. Next, the Bi-RNN output at each step is passed through a fully connected feedforward network with a sigmoid output layer, followed by a normalization step.

$$s_i = \sigma(W_2\, \sigma(W_1 o_i^q + b_1) + b_2)\;; \quad f^{\text{int}}(q_i|q) = s_i / \sum_{k=1}^{n} s_k \quad (2)$$

Exact pairwise matching. We define the exact pairwise matching (EPM) score for a <q, a> pair as a weighted sum of all the overlapped words (corresponding to the black squares of the exact pairwise matching module in Figure 2):

$$s^{\text{EPM}}(q, a) = \sum_{i=1}^{n} f^{\text{int}}(q_i|q)\, \mathbb{I}(q_i \in a) \quad (3)$$

Here $\mathbb{I}(q_i \in a)$ is an indicator function that returns 1 if q_i appears in a and 0 otherwise.

The objective of exact pairwise matching is to assign higher scores to correct answers than incorrect ones, and therefore guides f^{int} to put more weights on words that occur in the correct answers but not in other answer candidates. Specifically, we use a standard cross-entropy loss function, minimizing which encourages increasing s^{EPM} of the correct answer a^+ compared with a set of randomly sampled non-answers $\mathcal{A}^-$:

$$\mathcal{L}^{\text{EPM}} = -\log \frac{\exp(s^{\text{EPM}}(q, a^+))}{\sum_{a\in\{a^+\}\cup\mathcal{A}^-} \exp(s^{\text{EPM}}(q, a))} \quad (4)$$

Once the internal word re-weighting function f^{int} is learned, we can directly combine the predicted word weights of a question with an off-the-shelf *tf-idf* based IR [22], so that the importance of a word is measured by both its corpus-level statistical importance and the context-aware semantic importance. Specifically, the *tf-idf* weight of each question word in an answer candidate is multiplied with its f^{int} score, before they are summed up as the final score of the answer candidate. Despite being simple, as experiments will show, adopting this re-weighting (RW) strategy helps retrieve better results, which also outperforms some existing baselines.

4 EXTERNAL WORD ASSOCIATION

It is common that a question word semantically aligns with different words in a correct answer, e.g., "*weight*" in q with "*lb*" or "*weighs*" in a in Figure 2. Unfortunately, exact pairwise matching does not exploit such connections, and is therefore incapable of solving the lexical gap problem. This motivates us to design the soft pairwise matching objective to further explore the associations between a question word and other words that are outside the question but likely to appear in correct answers. At the end of this section, we discuss three different strategies that leverage the word associations to expand a question at retrieval time.

Soft pairwise matching. Analogous to the previous section, we define a soft pairwise matching (SPM) score, $s^{\text{SPM}}(q, a)$, based on word associations from both the question side and the answer side. Recall that in Equation 1, the question has already been encoded into $[o_1^q, o_2^q, ..., o_n^q]$. During training, we use the same Bi-RNN (with shared parameters) for questions to encode an answer ($a = [a_1, ..., a_m]$) as $[o_1^a, ..., o_m^a]$. Afterwards, we calculate the external word association score $f^{\text{ext}}(q_i, a_j|q, a)$ for every word pair <q_i, a_j> as the cosine similarity of their corresponding Bi-RNN outputs,

Algorithm 1: General Question Expansion

1 **Result:** Question expansion words and weights
2 **Input:** internal word weights $f^{\text{int}}(q_i|q)$; vocabulary V;
3 expansion similarity function $p(q_i, w)$;
4 hyper-parameters: integer N; float δ
5 **for** q_i *in question* $q = [q_1, \ldots, q_n]$ **do**
6 $p(q_i, q_i) = 0$
7 /* Select N words with highest expansion similarities: */
8 $\{w_{i,1}, \ldots, w_{i,N}\} = \text{top\_k}([p(q_i, w_0), \ldots, p(q_i, w_{|V|})], N)$
9 /* δ controls the contribution of expansion terms */
10 $s_{i,j} = f^{\text{int}}(q_i|q)\, p(q_i, w_j)\, \delta,\ j = 1, \ldots, N$
end
11 Sum $s_{i,j}$ that corresponds to the same word
12 **return** distinct $w_{i,j}$ with corresponding aggregated $s_{i,j}$

and aggregate them into the overall soft pairwise matching score $s^{\text{SPM}}(q, a)$ as follows:

$$f^{\text{ext}}(q_i, a_j|q, a) = \frac{(o_i^q)^T \cdot o_j^a}{|o_i^q| \ \ |o_j^a|} \tag{5}$$

$$s^{\text{SPM}}(q, a) = \sum_{i=1}^{n} f^{\text{int}}(q_i|q) \max_{1 \le j \le m} f^{\text{ext}}(q_i, a_j|q, a) \tag{6}$$

Here $f^{\text{int}}(q_i|q)$ is the internal word weight from Equation 2, representing the relative importance of the i-th word in the question. The intuition behind this SPM function is that an answer should achieve a high score if for each important word of the question, at least one word in the answer matches well with it. We also empirically test for each question word q_i, summing $f^{\text{ext}}(q_i, a_j|q, a)$ of all the answer words a_j instead of using max pooling, which leads to similar results but takes longer time to train. Same as in the previous section, the soft pairwise matching function can also be trained using negatively sampled QA pairs:

$$\mathcal{L}^{\text{SPM}} = -\log \frac{\exp(s^{\text{SPM}}(q, a^+))}{\sum_{a \in \{a^+\} \cup \mathcal{A}^-} \exp(s^{\text{SPM}}(q, a))} \tag{7}$$

Optimizing $\mathcal{L}^{\text{SPM}}$ alone can affect both internal word re-weighting (f^{int}) and external word association (f^{ext}) modules through back-propagation, but the effect on the former can be distant and indirect. Therefore, we propose to jointly optimize both objectives:

$$\min_{\Theta} (\mathcal{L}^{\text{EPM}} + \mathcal{L}^{\text{SPM}}) \tag{8}$$

Here, Θ represents all parameters including word embedding vectors, those in Bi-RNN, and $\{W_1, b_1, W_2, b_2\}$.

Question expansion based on external word associations. Although directly ranking answer candidates through soft pairwise matching seems applicable, it requires encoding all the review sentences with Bi-RNN at testing time, which is computationally expensive for PQA. We propose to use easy-to-compute word similarity functions to transform the neural external word associations from the training phase (Eq. 6) to a small set of expanded keywords for a question, which promotes interpretability as it is easier for humans to check expansion words than pairwise word associations and can also be used to enhance the performance of keyword-based IR.

There has been a broad spectrum of works on question/query expansion (QE) in the past for retrieving web pages [46], emails [23], answers [38] and so on. However, many existing QE techniques are not directly applicable in this PQA scenario (e.g., those trained with click-through data). The most related QE techniques to ours are those using word embeddings [12, 24, 49]. The difference of our QE methods is that we jointly consider internal word weights in training word associations (f^{int} in Eq.6), and find external relevant words not only at the word embedding level, but also at higher levels projected by the trained neural architecture. Algorithm 1 outlines a general query expansion scheme based on which we developed three intuitive variants (**QE1/2/3**) using different *expansion similarity functions* (p /p'/p''). Given an expansion similarity function, it finds N most similar words for each question word (line 8), and their weights are the product of the internal word weights f^{int}, the expansion similarities $p(q_i, w)$ and a hyper-parameter δ (line 9). Duplicated expansion words are aggregated by summing up their individual weights (line 11). The final output is a set of weighted external words, which can be combined with the re-weighted internal words and then input together to the off-the-shelf keyword-based IR method as discussed earlier.

QE1: Word embedding similarity. This is a basic expansion similarity function that directly leverages trained word embeddings [12]. It is defined as the cosine similarity between the word embeddings of a question word q_i and an arbitrary word w from the vocabulary:

$$p(q_i, w) = \frac{e(w)^T \cdot e(q_i)}{|e(w)| \ |e(q_i)|} \tag{9}$$

Word embeddings in our model are initialized with pre-trained results [33] in the general domain, but further trained to be product domain specific [12] and can be more suitable for query expansion in our settings.

QE2: Higher-level word similarity. During training, the raw word embeddings of a sentence are further encoded by the Bi-RNN, which outputs higher-level representations at each step. We hypothesize that even without context (i.e., surrounding words), such low to high level transformations can help capture word associations more effectively. Therefore, we propose to leverage the learned parameters within the GRU cell. Specifically, we concatenate the word embedding $e(q_i)$ (same for $e(w)$) with zero vectors in both forward and backward directions, and project them by part of the GRU parameters. The projected vectors are concatenated into higher-level representations $e'(q_i)$ and $e'(w)$ and measured by cosine similarity:

$$\begin{aligned} e'(q_i) &= [\phi(\vec{W}[e(q_i), 0] + \vec{b}),\ \phi(\overleftarrow{W}[e(q_i), 0] + \overleftarrow{b})] \\ e'(w) &= [\phi(\vec{W}[e(w), 0] + \vec{b}),\ \phi(\overleftarrow{W}[e(w), 0] + \overleftarrow{b})] \\ p'(q_i, w) &= \frac{e'(w)^T \cdot e'(q_i)}{|e'(w)| \ |e'(q_i)|} \end{aligned} \tag{10}$$

QE3: Context-aware similarity. We further propose a question-context-aware QE strategy, which considers the entire question sentence at testing time and dynamically calculates expansion word similarities based on its Bi-RNN encodings (note for computational efficiency, there is no context considered for the expansion word side). Formally, with o_i^q being the Bi-RNN output corresponding to the i-th question word, and $e'(w)$ the same as in Equation 10, the new expansion similarity is defined as:

$$p''(q_i, w) = \frac{e'(w)^T \cdot o_i^q}{|e'(w)| \ |o_i^q|} \tag{11}$$

Comparing the three variants, QE1 and QE2 are off-line strategies meaning that the expansion similarity between any two words in the vocabulary can be pre-calculated, stored in a table, and ready to look up at testing time. QE2 is slightly more complex than QE1 as it requires to project word embeddings. They both enjoy a higher computational efficiency than QE3, which is an online strategy that needs to dynamically compute the expansion similarities based on the context of a specific question. However, QE3 is closer to how the external word association score f^{ext} is optimized during training, and can potentially achieve the best performance.

5 EXPERIMENTS

Due to the lack of labeled review sentences (being relevant/irrelevant to a question), previous works [30, 43] adopted a *synthetic* evaluation setting using large-scale historical QA pairs. Specifically, answers to all questions are gathered as an entire answer candidate pool for retrieval. For each question, its paired answer given by a human expert (the top rated one if there are many), is treated as correct whereas all other answer candidates as incorrect. The candidate pools for training, dev and testing sets are separated. In addition to this synthetic setting, we also experiment under a *realistic* scenario where we retrieve review sentences to answer questions, and ask human annotators to evaluate their relevance.

5.1 Experimental Setup

Datasets. We use the Amazon QA [30] and review datasets [29] from 5 different departments. We pre-process these raw data slightly differently from [30] and the statistics are summarized in Table 1. First, we do not classify a question as *yes-no* or *open-ended* type. As emphasized in [30], even for yes-no questions, it is critical to find relevant review sentences as supporting evidence. Therefore, we directly treat PQA as a review sentence selection task. Second, we filter out QA pairs whose answers contain less than two words other than stop words or "*yes*"/"*no*", because they contain little useful information for either training or testing. The last column of Table 1 shows that after filtering, we still keep 80%∼97% of all the questions in the raw datasets, much higher than the 44% classified as open-ended in [30], additionally showing that treating PQA as a retrieval problem has a broader coverage. The datasets are randomly divided into train, dev and test set using 7: 1: 2 ratio. The same pre-processed data splits are used for all methods.

Evaluation Measure. For synthetic evaluation, we follow previous work [30, 43] to use average Area Under the Curve (AUC):

$$\text{AUC} = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}}\frac{1}{|\mathcal{A}|-1}\sum_{a^-\in\mathcal{A}-\{a^+\}}\mathbb{I}(\text{score}(q,a^+)>\text{score}(q,a^-)),$$

where $\mathcal{Q}$ represents all the questions in dev or test set, and $\mathcal{A}$ is the corresponding set of all the answers. $\mathbb{I}(\text{score}(q,a^+)>\text{score}(q,a^-))$ is a binary indicator function that returns 1 when the paired answer is ranker higher than the non-answer[1]. In the real user study experiment, where humans are asked to annotate the relevance of each retrieved review sentence, we use normalized discounted cumulative gain (nDCG), a popular ranking metric for IR [21].

[1]Same as previous work [30, 43], we approximately assume only the paired answer is correct to a question. AUC essentially measures the percentage of non-paired answers getting lower scores than the paired one, and is less sensitive to the dataset noise compared with other ranking metrics like Mean Reciprocal Rank (MRR) or nDCG, especially when the number of correct answers to a question is significantly smaller than the answer pool size $|\mathcal{A}|$.

Dept.	vocab size	# of QA pairs			subjective
		Train	Dev	Test	Qs (%)
Appliances	31,694	6,156	879	1,760	97.17
Baby	62,267	14,901	2,128	4,259	73.58
Patio	89,449	37,427	5,346	10,695	89.72
Tools	116,635	62,242	8,891	17,785	87.96
Electronics	100,000	179,858	25,694	51,389	81.76

Table 1: Dataset statistics.

Baseline methods. The baseline methods can be divided into those without and with explicit question expansion. Those without explicit QE are: (1) **Moqa** and **BM25+** [30]: They both use the same mixture-of-experts framework to jointly optimize answer-review and question-review relevance functions. Moqa models the relevance functions as approximated bilinear models to match sentence pairs in latent vector spaces, while BM25+ models them as Okapi BM25+ ranking function with a few learnable parameters [27]. We used their published code for both Moqa and BM25+ on our pre-processed datasets. (3) **OQ** and **OQ - stop**: Two naive baselines using the same *tf-idf* IR module as in our framework, one directly using the original question to query and the other with stopwords removed from the question using SpaCy. (4) **RW**: Our internal word re-weighting method as described in Section 3.

We select the following baselines with question expansion: (5)**PRF-TFIDF**: A classic QE method [6] that expands a query using the top-K *tf-idf* words from the top-N pseudo-relevant sentences returned by IR. (6) **PRF-RM**: A popular QE strategy based on relevance model [25]. Specifically, given original query q_0, the probability of selecting a term t in the reformulated query is: $P(t|q_0) = (1-\lambda)P'(t|q_0)+\lambda\sum_{d\in D_0}1/|D_0|P(t|d)P(q_0|d)$, where D_0 is the set of top-N pseudo-relevant documents returned by q_0 and λ is set at 0.5 following previous work. $P(t|q_0)$ is defined as the normalized *tf-idf* weight of t in q_0 and $P(t|d)$ the add-one smoothed language model: $P(t|d) \propto \text{tf}(t,d)+1$. We select top-K terms with highest probability to expand query q_0. (7) **GloVe** and **GloVe$^{\text{tgt}}$**: Following state-of-the-art word embedding based QE works [12, 49], question words are expanded through similarities (equivalent to $p(q,w_i)$ in Equation 9) measured by two types of word embeddings: (i) those pre-trained by GloVe [33] and (ii) those fine-tuned on our target (tgt) domain corpus with the relevance-based objective designed by [49] for search tasks. Specifically, the QE process is equivalent to Algorithm 1 but without considering learned internal word weights (i.e., fix f^{int} to 1 in line 10). (8) **GloVe+RW** and **GloVe$^{\text{tgt}}$+RW**: Same as above but further take into account our learnt internal word weights (i.e., f^{int} given by Riker in line 10). (9) **Riker+QE1/QE2/QE3**: After Riker is trained, we test the three question expansion variants described in Section 4 respectively.

5.2 Results

Comparing with baselines without QE. The upper half of Table 2 compares all the methods without explicit question expansion. We observe that our internal word re-weighting (RW) strategy outperforms the search baselines OQ and OQ - stop, showing the benefit of dynamically predicting word weights over naive strategies using equal weights or removing stop words (which may fail

	Appliances		Baby		Patio		Tools		Electronics	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
BM25+	56.577	56.746	65.799	65.385	65.349	64.497	64.280	65.219	63.913	63.818
OQ	78.120	79.039	87.078	87.600	85.020	84.510	86.131	86.049	86.356	86.408
OQ - Stop	77.448	78.319	86.002	85.963	83.559	83.311	84.873	84.888	84.537	84.683
Moqa	76.367	76.314	83.836	83.349	85.292	85.011	86.673	86.421	87.836	87.824
RW	78.332	79.653	87.743	89.043	85.204	84.792	86.524	86.436	87.063	87.082
PRF-TFIDF	74.382	75.061	84.802	85.102	82.205	82.884	83.534	83.723	82.847	83.492
PRF-RM	78.446	79.801	87.931	88.235	85.469	85.113	86.766	86.688	86.058	86.083
GloVe	65.410	66.918	74.775	74.456	73.760	73.440	77.785	77.676	78.072	78.003
GloVe^{tgt}	76.146	77.293	85.777	85.517	82.748	82.590	84.613	84.442	83.126	83.054
GloVe+RW	77.071	78.628	88.600	88.307	86.147	85.599	87.261	87.156	86.692	86.749
GloVe^{tgt}+RW	79.959	80.981	90.298	90.053	87.884	87.856	89.060	88.918	88.151	88.126
Riker+QE1	78.733	79.920	89.486	89.170	86.744	86.384	86.436	87.836	87.380	87.436
Riker+QE2	80.405	81.427	91.527	91.367	89.427	89.681	90.394	90.175	89.493	89.537
Riker+QE3	**84.415**	**84.629**	**92.262**	**92.086**	**91.365**	**91.368**	**91.969**	**91.643**	**91.095**	**91.010**

Table 2: PQA results (AUC) under the synthetic setting. The upper(lower) half shows methods without(with) explicit QE. Riker+QE2/QE3 achieve the best overall performance, showing that it mines effective keyword representations.

because words such as "*without*", "*last*", "*bottom*" that are often treated as stop words are actually important for product-related questions). RW also achieves comparable or better performances to the end2end Moqa baseline especially when the dataset size gets smaller, indicating that the task to predict word importance is less sensitive to dataset sizes, and is especially suitable for new domains without much available data. Another observation is that with our pre-processing steps, using the original question (OQ)[2] to search has already achieved performances close to Moqa, showing that keyword-based search can be decently effective in product domain. Next, we will discuss that RW also plays a key role for QE to succeed.

Comparing Riker with baselines with QE. The lower half of Table 2 compares all methods with explicit question expansion. Our proposed Riker+QE2/QE3 variants outperform all baselines. GloVe$^{(tgt)}$+RW using word embeddings fine-tuned for retrieval purpose in the target domain corpus is the strongest baseline, but is less effective than some Riker variants because we jointly trained internal word re-weighting and external word association in a unified framework. Notice that word embedding based QE baselines perform poorly if not combined with RW (GloVe$^{(tgt)}$), again showing the importance of internal word re-weighting for PQA because treating question words equally when incorporating external terms will introduce much noise. The classic QE methods PRF-TFIDF and PRF-RM generally works not as well as embedding based ones because the representation power of word embedding can be more expressive than corpus level statistics.

Comparing Riker variants. Riker+QE2 with the novel use of GRU cell parameters outperforms Riker+QE1, demonstrating that the trained neural layer can project word embeddings into more effective higher-level semantic representations. As expected, Riker+QE3, which takes the context information of an entire question into account, performs the best, but it comes at a cost of computing vector similarities at testing time as discussed earlier. In practice, one may consider the trade-off between efficiency and effectiveness when choosing from QE2 and QE3.

[2]This baseline was not tested in [30].

Objective	dev AUC		test AUC	
	RW	QE3	RW	QE3
EPM	87.434	86.563	**88.117**	86.632
SPM	87.104	91.646	87.589	91.426
EPM+SPM	**87.743**	**92.626**	88.041	**92.086**

Table 3: Training objective ablation.

Ablation study of training objectives. Table 3 shows the effect of our proposed training objectives. Due to space limit, we only show the Baby dataset but the results for others are similar. Training using only the EPM objective gets comparable performance as using both objectives for our RW method, but makes QE3 ineffective because without SPM, the Bi-RNN does not learn how to associate words at the high level. The QE3 strategy works the best when the EPM (Eq. 4) and SPM (Eq. 7) objective are optimized simultaneously, showing that with explicit supervision for internal word re-weighting using EPM, Riker can better find the important expansion words to be associated with a question.

Parameter sensitivity. Figure 3 shows that Riker+QE3, our best variant, is insensitive to QE hyper-parameters N and δ on Baby dataset. Similar results are observed on the other datasets. Riker+QE3 significantly outperforms the baseline without QE (i.e., RW in Table 2), whose AUC for the dev/test set is 87.743/89.043, far below the current coordinate ranges in Figure 3. Results on dev and test datasets are similar, with AUCs reaching the best when the expansion scale $\delta = 0.15$. The performance tends to get better as N increases, but involving more and more external words will make keyword-based IR less understandable, and hence we balance effectiveness and interpretability by upper-bounding N at 70.

Real User Evaluation So far, we have been experimenting under the *synthetic* setting, where Riker performs the best at selecting answer sentences. However, one might suspect such performance gain could come from that Riker is trained to optimize answer ranking, whereas the existing state-of-the-art PQA method Moqa [30] jointly models the relevance between questions, answers and reviews in a comprehensive probabilistic model.[3]

[3]It is fair to compare Riker with other baselines such as GloVe^{tgt}(+RW) under the synthetic evaluation, since they all directly match QA pairs, not through reviews.

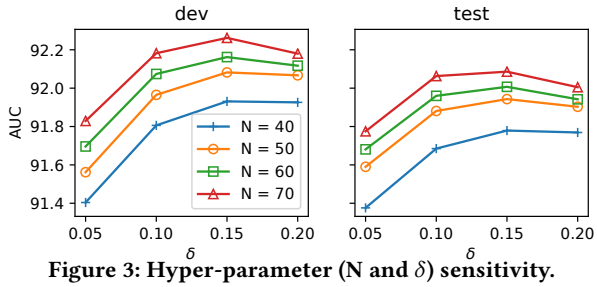


Figure 3: Hyper-parameter (N and δ) sensitivity.

Dept.	nDCG@10 (%)								Fleiss' Kappa
	User1		User2		User3		**Avg**		
	Moqa	QE3	Moqa	QE3	Moqa	QE3	Moqa	QE3	
Appl.	18.60	55.22	42.36	56.60	20.74	43.67	27.24	**51.83**	0.335
Baby	26.38	73.97	25.50	66.29	14.47	54.13	23.78	**64.80**	0.459
Patio	23.89	61.80	26.53	49.30	24.68	56.64	25.03	**55.91**	0.293
Tools	25.03	49.67	26.62	42.15	33.49	43.55	28.38	**45.12**	0.392
Elec.	27.51	59.57	35.49	53.42	21.81	58.47	28.27	**57.15**	0.370

Table 4: Real user evaluation of Moqa and RIKER+QE3.

In order to show RIKER's generalization ability to selecting review sentences, we conduct a real user study where different methods retrieve *review sentences* corresponding to the product to answer questions. We randomly sampled 200 questions, 40 from each department, and there are on average 474 review sentences as answer candidates for each question.[4] For every retrieved sentence, three different human annotators were asked to score it with 2, 1 or 0, corresponding to being *relevant*, *partly relevant* and *irrelevant*.[5] The results are summarized in Table 4. In this realistic setting, RIKER+QE3 still substantially outperforms Moqa, indicating that using RIKER predicted keyword representations of a question can also search product reviews more effectively than latent semantic representation approaches. This is partly because review sentences often contain more personal experience as well as sentiment-related words/phrases than human provided answers in our training sets, and their sentence-level semantics can be hard to match with a question. However, once the keywords associated with a question are predicted, relevant review sentences can be more easily retrieved using keyword-based IR. Fleiss' Kappa scores [15] are used to measure the consistency among different users' labels, which show fair and moderate agreements (>0.35) for most departments.

6 DISCUSSION

Section 5 shows that keyword representations mined by RIKER can enhance the performance of keyword-based IR, in comparison with various existing query expansion and PQA methods, leading to a more effective PQA framework with the ranking process still interpretable based on matched keywords. In this section, we will further compare RIKER with attention mechanisms[9, 31, 41, 42], which have recently been widely used in DNNs for sentence pair modeling and can provide interpretations of how two sentences are matched by revealing word-level associations. As we need to first clarify how to apply the attention mechanisms to our task, we separate the discussion from other methods in Section 5.

6.1 An Attention-based Deep QA Model

We briefly show how to adapt an end2end DNN QA model to the PQA setting. We choose a standard word pairwise attention mechanism [31] that are proved effective in many recent sentence pair modeling works [9, 41, 42] . For fair comparison, we use the exact same Bi-RNN architecture as in RIKER to encode the question/answer sentences and compute attention from its outputs $[o_1^q, ..., o_n^q]$ and $[o_1^a, ..., o_m^a]$. The final matching score is S^{DNN}:

$$e_{i,j} = F(o_i^q)^T \cdot F(o_j^a) \quad 1 \le i \le n,\ 1 \le j \le m \tag{12}$$

$$\bar{o}_i^q = \sum_{j=1}^{m} \frac{\exp(e_{i,j})}{\sum_{k=1}^{m} \exp(e_{i,k})} o_j^a, \quad \bar{o}_j^a = \sum_{i=1}^{n} \frac{\exp(e_{i,j})}{\sum_{k=1}^{n} \exp(e_{k,j})} o_i^q \tag{13}$$

$$v^q = \sum_{i=1}^{n} G([o_i^q, \bar{o}_i^q]), \quad v^a = \sum_{j=1}^{m} G([o_j^a, \bar{o}_j^a]) \tag{14}$$

$$s^{\text{DNN}} = H([v^q, v^a]) \tag{15}$$

Here, F, G and H are different feedforward networks and are all instantiated with one hidden layer in our experiments. The same objective function (Eq. 4 or 7 but using s^{DNN}), regularization and training procedure as RIKER is used to train this DNN model.

Using attention for question expansion We compare the learned attention from this DNN model head-to-head with RIKER's word associations for question expansion. Specifically, we adapt the same QE2 and QE3 strategy in Section 4 for the attention mechanism, with the word similarity functions changed to:

$$p'^{\text{ATTN}}(q_i, w) = F(e'(w))^T F(e'(q_i))/(|F(e'(w))|\ |F(e'(q_i))|) \tag{16}$$

$$p''^{\text{ATTN}}(q_i, w) = F(e'(w))^T F(o_i^q)/(|F(e'(w))|\ |F(o_i^q)|) \tag{17}$$

Here F is the trained feedforward net in Equation 12. We denote these two baselines as **ATTN+QE2** and **ATTN+QE3**.

6.2 Case Studies

We first show some qualitative results to help illustrate the interpretability of RIKER's word associations in comparison with that of DNN. *How to evaluate interpretability is still an open issue, but intuitively in the PQA scenario, a method that comes up with more relevant keywords and uses them to retrieve better answer sentences is more interpretable than one with less relevant words and finds less useful answers.* Table 5 demonstrated using QE3 strategy to expand several common questions in the largest Electronics dataset. In general, RIKER's expansion words appear to be better. For example, for the last question about turning off the alarm, RIKER+QE3 suggests words such as"*alert*", "*disarm*", "*deactivate*", while ATTN+QE3's result seems less relevant. Due to space limit, we omit examples from other datasets and the qualitative comparison between RIKER+QE2 and ATTN+QE2, but we observed the same phenomenon that RIKER can associate more relevant words.

We next quantitatively compare the keyword representations mined by RIKER and those by attention mechanisms in terms of how much they help boost the *tf-idf* based search framework. This evaluation method is also the same as the so-called *functionally-grounded*

[4]The average number of reviews sentences for each product in the datasets: Appliance–342.88, Baby–720.87, Patio–319.96, Tools–297.92, Electronics–686.23.
[5]We adopt these three scores since annotators found it hard to label many review sentences as definitely relevant or not, due to the subjectivity of PQA.

Question	ATTN+QE3	Riker+QE3
"can i use this for outdoors? thank..."	sensitive outside weather aviation camping d100 cop extensively sonar 60csx eave coordinate 1980 beltronics fahrenheit resonate ra dingy enemy 9500ix	outside outdoor backyard camping weather windy unprotected rain indoor rv camper atmosphere weatherproof hiking rooftop wherever desert
"how large be the base of the light stand? i need to know if these stand can fit in a smallish space."	ceiling riser plywood sanus symmetrical perpendicular shelf finesse firing endure attached peerless 50-inch cushioning sleeker stretchy 15inch sail	folding standing spaced headboard collapse stool heel pray centimeter small raise hang 161 hanging chair bookcase strong ledge pedestal liking
"how do you turn the alarm off?"	switched hardwired goodbye elevator reseat instantaneously tutorial uninterrupted shutdown shaft hitch shutoff swiping uninstalled direction quadrant	alert disarm deactivate beeping buzzer countdown unplug regulate annoying beep inactive disable checkbox snooze timer shutoff inactivity

Table 5: Qualitative comparison of word expansion with QE3. In general, the top few words expanded by Riker+QE3 (e.g., "*outside*", "*folding*", "*alert*") is more relevant than those using DNN attentions (e.g., "*sensitive*", "*ceiling*", "*switched*").

evaluation methodology suggested by [13] for interpretable machine learning, which advocates using the performance improvement of some interpretable model (i.e., the *tf-idf* based search in our setting) as proxy to evaluate the explanation/interpretation quality. We prefer this evaluation method to user studies (e.g., hiring humans to score the expanded keywords such as those in Table 5) in that it is more objective, much cheaper and easier to conduct. For example, it is not easy for humans to compare the two columns in Table 5 at large scale, especially when both contain some relevant keywords.

Table 6 summarizes the comparison between Riker and ATTN with both QE2 and QE3. We also list the performance of end2end DNN at the bottom despite its known lack of interpretability. As we can see, all methods outperform the RW baseline, indicating that the learned pairwise attentions from DNN indeed associate interpretable expansion words. Consistent with Riker's results, ATTN+QE2 is less effective than ATTN+QE3. Importantly, both ATTN+QE2 and ATTN+QE3 are inferior to their counterparts using Riker, showing that our methods can generate more interpretable expansion words. The reason is that the attention within DNN is trained only as intermediate weights used for aggregating latent word representations (Eq. 13), while our training objective directly optimize the score function s^{SPM}, which is the sum of pairwise word associations (Eq. 6). The end2end DNN model achieves the best performance when the dataset size is large, but its interpretability is worse than ATTN+QE2/QE3 as the last few steps (Eq. 14 and 15) are non-linear and the low-dimensional dense feature vectors still need further interpretation [14, 20]. *Our framework can be viewed as a model that constrains the question and review sentences to be matched in the lexical space with sparse bag-of-word features, and is optimized to strike a good balance between effectiveness and interpretability.*

7 RELATED WORK

(Product) answer sentence selection. Product QA on large-scale Amazon datasets is first studied in [30], which proposes a mixture-of-experts framework to jointly model review-answer and review-question relevance with latent vector representations. Wan et al.

	Appl.	Baby	Patio	Tools	Elec.
RW	79.653	89.043	84.792	86.436	87.082
Riker+QE2	81.427	91.367	89.681	90.175	89.537
ATTN+QE2	78.405	89.434	86.965	88.749	88.112
Riker+QE3	**84.629**	**92.086**	91.368	91.643	91.010
ATTN+QE3	80.682	90.070	87.729	89.428	88.587
DNN (end2end)	82.115	90.318	**91.756**	**94.158**	**95.927**

Table 6: Quantitative comparison of Riker, end2end DNN and using its ATTeNtion for QE on test set. For methods using keyword-based search (except end2end DNN), higher retrieval performance (AUC) corresponds to better interpretability reflected by the *tf-idf* proxy [13].

[43] build upon the above framework to handle the situation where questions can have multiple answers and reviews can be subjective by including more features (e.g., reviewer expertise and biases). Similar to [30], our framework currently models text information only and optimizes one correct answer, but can be extended to the scenario in [43] by averaging the scores of all correct answers and incorporating non-text features in our objective functions. Yu et al. [48] focus on "yes-no" questions only, and shows that learning latent product aspects and aspect-specific embeddings can help make binary predictions. Outside the product domain, answer sentence selection (e.g., [41, 42]) has been a popular topic in general. Our internal word re-weighting function and soft pairwise matching module are related to attention mechanism (e.g., [50]) and pairwise semantic interactions (e.g., [31, 41]), which are two popular techniques applied in general answer sentence selection models, but differ from them in the sense that we use the explicit weights (f^{int} and f^{ext}) to directly rank answers, while their techniques are integrated in latent vector representations in a less interpretable end2end fashion.

Query expansion. Query expansion aims to automatically expand a query with additional terms to get better search results, and has been a longstanding topic [6, 23, 25, 38, 46]. Question expansion strategies in our work (especially QE1) are most related to recent word embedding based techniques such as [24]. Diaz et al. [12] further show that fine-tuning word embeddings on domain specific corpus can help get better performance, but it requires re-training word embeddings for every query. Zamani et al. [49] propose learning offline word embeddings based on "relevance" instead of "proximity". In this work, we do not focus on developing novel QE strategies, and instead apply existing or most intuitive ones to employ the learned word weights and associations by Riker and test its effectiveness. It is interesting for future work to see whether QE techniques learned from other data sources may be combined with Riker to achieve overall better performances.

Interpretable machine learning. Interpretable machine learning receives a lot of attention recently in a broad range of fields [16–18]. Some works focus on understanding general machine learning algorithms including deep neural networks, e.g., through proxy models [37], salient mapping [3], interpreting latent semantic representations [5], or adversarial networks [10]. On the application level, people have explored making systems more interpretable to users including recommendation systems [2], visual QA [32], multiple choice answers [40] and so on. To the best of our knowledge,

we make the first effort towards interpretable PQA and advocate mining keyword, rather than vector, representations of a question to boost the effectiveness of keyword-bases search.

8 CONCLUSION

This work proposes a new hybrid framework combining the advantages of deep models and keyword-based search towards effective yet interpretable PQA. We employ an easily interpretable *tf-idf* based IR module to rank answers, but in order to address the lexical gap problem, we propose Riker to mine rich keyword representations for customer questions, consisting of re-weighted internal words and associated external words. Experimental results show that the mined keyword representations can help improve PQA performance substantially over existing PQA methods, while at the same time preserve good interpretability of the keyword-based search paradigm.

ACKNOWLEDGMENTS

This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, and Ohio Supercomputer Center [8]. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al. 2016. Tensorflow: a system for large-scale machine learning. In *OSDI*, Vol. 16. 265–283.
[2] Q. Ai, V. Azizi, X. Chen, and Y. Zhang. 2018. Learning Heterogeneous Knowledge Base Embeddings for Explainable Recommendation. *arXiv preprint arXiv:1805.03352* (2018).
[3] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. R. Müller, and W. Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10, 7 (2015), e0130140.
[4] R. Baeza-Yates, B. Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463.
[5] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. 2017. Network dissection: Quantifying interpretability of deep visual representations. *arXiv preprint arXiv:1704.05796* (2017).
[6] C. Buckley, G. Salton, J. Allan, and A. Singhal. 1995. Automatic query expansion using SMART: TREC 3. *NIST special publication sp* (1995), 69–69.
[7] D. Carmel, L. Lewin-Eytan, and Y. Maarek. 2018. Product Question Answering Using Customer Generated Content-Research Challenges. In *SIGIR*. 1349–1350.
[8] Ohio Supercomputer Center. 1987. Ohio Supercomputer Center. http://osc.edu/ark:/19495/f5s1ph73
[9] Q. Chen, X. Zhu, Z. Ling, S. Wei, H. Jiang, and D. Inkpen. 2016. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038* (2016).
[10] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*. 2172–2180.
[11] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
[12] F. Diaz, B. Mitra, and N. Craswell. 2016. Query expansion with locally-trained word embeddings. *arXiv preprint arXiv:1605.07891* (2016).
[13] F. Doshi-Velez and B. Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* (2017).
[14] M. Du, N. Liu, and X. Hu. 2018. Techniques for interpretable machine learning. *arXiv preprint arXiv:1808.00033* (2018).
[15] J. L. Fleiss and J. Cohen. 1973. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and psychological measurement* 33, 3 (1973), 613–619.
[16] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. 2018. Explaining Explanations: An Approach to Evaluating Interpretability of Machine Learning. *arXiv preprint arXiv:1806.00069* (2018).
[17] B. Goodman and S. Flaxman. 2016. European Union regulations on algorithmic decision-making and a" right to explanation". *arXiv preprint arXiv:1606.08813* (2016).
[18] D. Gunning. 2017. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web* (2017).
[19] Matthew H. and Ines M. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. *To appear* (2017).
[20] S. Jain and B. C. Wallace. 2019. Attention is not Explanation. *arXiv preprint arXiv:1902.10186* (2019).
[21] K. Järvelin and J. Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM TOIS* 20, 4 (2002), 422–446.
[22] K S. Jones, S. Walker, and S. E. Robertson. 2000. A probabilistic model of information retrieval: development and comparative experiments: Part 2. *Information processing & management* 36, 6 (2000), 809–840.
[23] S. Kuzi, D. Carmel, A. Libov, and A. Raviv. 2017. Query Expansion for Email Search. In *SIGIR*. 849–852.
[24] S. Kuzi, A. Shtok, and O. Kurland. 2016. Query expansion using word embeddings. In *CIKM*. 1929–1932.
[25] V. Lavrenko and W. B. Croft. 2001. Relevance Based Language Models. In *SIGIR*.
[26] Z. C. Lipton. 2016. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490* (2016).
[27] Y. Lv and C. Zhai. 2011. Lower-bounding term frequency normalization. In *CIKM*. 7–16.
[28] G. Marcus. 2018. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631* (2018).
[29] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*. 43–52.
[30] J. McAuley and A. Yang. 2016. Addressing complex and subjective product-related queries with customer reviews. In *WWW*. 625–635.
[31] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit. 2016. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933* (2016).
[32] D. H. Park, L. A. Hendricks, Z. Akata, A. Rohrbach, B. Schiele, T. Darrell, and M. Rohrbach. 2018. Multimodal Explanations: Justifying Decisions and Pointing to the Evidence. In *CVPR*.
[33] J. Pennington, R. Socher, and C. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*. 1532–1543.
[34] J. Pérez-Iglesias, J. Pérez-Agüera, V. Fresno, and Y. Z. Feinstein. 2009. Integrating the probabilistic models BM25/BM25F into Lucene. *arXiv preprint arXiv:0911.5046* (2009).
[35] R. Pryzant, S. Basu, and K. Sone. 2018. Interpretable Neural Architectures for Attributing an Ad's Performance to its Writing Style. In *EMNLP Workshop BlackboxNLP*. 125–135.
[36] R. Řehůřek and P. Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *LREC Workshop on New Challenges for NLP Frameworks*. 45–50.
[37] M. T. Ribeiro, S. Singh, and C. Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *KDD*. 1135–1144.
[38] S. Riezler, A. Vasserman, I. Tsochantaridis, V. Mittal, and Y. Liu. 2007. Statistical machine translation for query expansion in answer retrieval. In *ACL*. 464–471.
[39] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.. In *ICCV*. 618–626.
[40] R. Sharp, M. Surdeanu, P. Jansen, M. A. Valenzuela-Escárcega, P. Clark, and M. Hammond. 2017. Tell me why: Using question answering as distant supervision for answer justification. In *CoNLL*. 69–79.
[41] G. Shen, Y. Yang, and Z. H. Deng. 2017. Inter-Weighted Alignment Network for Sentence Pair Modeling. In *EMNLP*. 1190–1200.
[42] Y. Tay, L. A. Tuan, and S. C. Hui. 2018. Multi-Cast Attention Networks for Retrieval-based Question Answering and Response Prediction. *arXiv preprint arXiv:1806.00778* (2018).
[43] M. Wan and J. McAuley. 2016. Modeling ambiguity, subjectivity, and diverging viewpoints in opinion question answering systems. In *ICDM*. 489–498.
[44] F. Wang and C. Rudin. 2015. Falling rule lists. In *AISTATS*.
[45] M. Wang, N. A. Smith, and T. Mitamura. 2007. What is the Jeopardy model? A quasi-synchronous grammar for QA. In *EMNLP-CoNLL*.
[46] J. Xu and W. B. Croft. 1996. Query expansion using local and global document analysis. In *SIGIR*. 4–11.
[47] Y. Yang, W. T. Yih, and C. Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *EMNLP*. 2013–2018.
[48] Q. Yu and W. Lam. 2018. Aware Answer Prediction for Product-Related Questions Incorporating Aspects. In *WSDM*. 691–699.
[49] H. Zamani and W. B. Croft. 2017. Relevance-based Word Embedding. In *SIGIR*. 505–514.
[50] X. Zhang, S. Li, L. Sha, and H. Wang. 2017. Attentive Interactive Neural Networks for Answer Selection in Community Question Answering.. In *AAAI*. 3525–3531.
[51] J. Zhao, Y. Su, Z. Guan, and H. Sun. 2017. An End-to-End Deep Framework for Answer Triggering with a Novel Group-Level Objective. In *EMNLP*.

A IMPLEMENTATION DETAILS

Some extra implementation details of this work are clarified here. For data pre-processing, we use SpaCy [19] to lemmatize and lowercase words. The vocabulary (Table 1) is constructed by including all words from the training set that either appear in the GloVe[33] pre-trained vocabulary or appear more than 5 times, except for the Electronics department, where for efficiency reasons we further truncate the vocabulary size to 100K from more than 300K words obtained by the above preprocessing method.

For the question expansion baselines PRF-TFIDF and PRF-RM (Table 2), we grid search the pseudo-relevant sentence number N from [10, 20, ..., 60] and top expansion word number K from [10, 20, ..., 80]. For all other QE baselines using Algorithm 1, we grid search N from [40, 50, 60, 70] and δ from [0.05, 0.10, 0.15, 0.20]. We select the best combination for each dataset based on dev set. For keeping the same level of interpretability, the expanded queries are evaluated through the same *tf-idf* based IR across our experiments, which is different from the language model based IR used in state-of-the-art word embedding based QE works [12, 49].

In the evaluation of the end2end DNN baseline (Table 6), because of the high computational cost to encode all answer candidates through Bi-RNN, we approximate its AUC performance by randomly sampling 1000 negative answers for each question following the method used in [30].

We use Tensorflow [1] to implement Riker. For each training epoch, we randomly sample 5 non-answers from the train/dev answer pool for each QA pair. We tune the model hyper-parameters based on the Baby domain dev set because of its moderate size for efficiency concerns, and use them for all other domains. We use the Adam optimizer with the learning rate set at 5e-4 and batch size set at 64. We add L2 regularization with coefficient 1e-4. For the keyword-based IR module, we employ an off-the-shelf inverted index based IR tool [36] and use the standard BM25 function with default parameters [34] to rank answers or review sentences. Source code and data will be available at: https://github.com/jiez-osu/PQA.

SurfCon: Synonym Discovery on Privacy-Aware Clinical Data

Zhen Wang*, Xiang Yue*, Soheil Moosavinasab†, Yungui Huang†, Simon Lin†, Huan Sun*
*The Ohio State University
{wang.9215,yue.149,sun.397}@osu.edu
†Abigail Wexner Research Institute at Nationwide Children's Hospital
{SeyedSoheil.Moosavinasab,Yungui.Huang,Simon.Lin}@nationwidechildrens.org

ABSTRACT

Unstructured clinical texts contain rich health-related information. To better utilize the knowledge buried in clinical texts, discovering synonyms for a medical query term has become an important task. Recent automatic synonym discovery methods leveraging raw text information have been developed. However, to preserve patient privacy and security, it is usually quite difficult to get access to large-scale raw clinical texts. In this paper, we study a new setting named *synonym discovery on privacy-aware clinical data* (i.e., medical terms extracted from the clinical texts and their aggregated co-occurrence counts, without raw clinical texts). To solve the problem, we propose a new framework SurfCon that leverages two important types of information in the privacy-aware clinical data, i.e., the *surface form information*, and the *global context information* for synonym discovery. In particular, the surface form module enables us to detect synonyms that look similar while the global context module plays a complementary role to discover synonyms that are semantically similar but in different surface forms, and both allow us to deal with the OOV query issue (i.e., when the query is not found in the given data). We conduct extensive experiments and case studies on publicly available privacy-aware clinical data, and show that SurfCon can outperform strong baseline methods by large margins under various settings.

KEYWORDS

Synonym Discovery, Privacy-Aware Clinical Data, Medical Term Recommendation

ACM Reference Format:
Zhen Wang*, Xiang Yue*, Soheil Moosavinasab†, Yungui Huang†, Simon Lin†, Huan Sun*. 2019. SurfCon: Synonym Discovery on Privacy-Aware Clinical Data . In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330894

1 INTRODUCTION

Clinical texts in Electronic Medical Records (EMRs) are enriched with valuable information including patient-centered narratives, patient-clinician interactions and disease treatment outcomes, which can be especially helpful for future decision making. To extract knowledge from unstructured clinical texts, synonym discovery [37] is an important task which can benefit many downstream applications. For example, when a physician issues a query term (e.g., "vitamin C") to find relevant clinical documents, automatically discovering its synonyms (e.g., "c vitamin", "vit c", "ascorbic acid") or even commonly misspelled variations (e.g. "viatmin c") can help to expand the query and thereby enhance the retrieval performance.

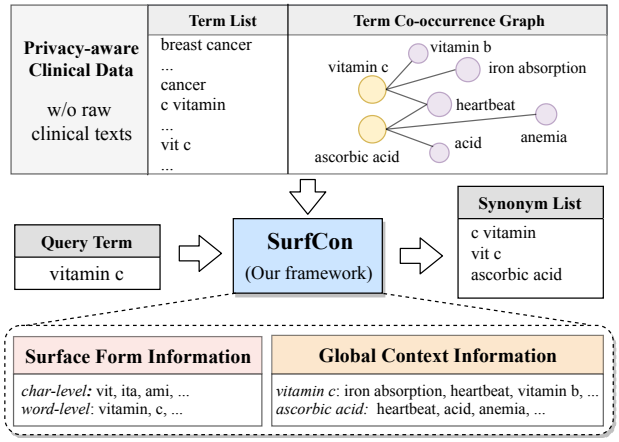


Figure 1: Task illustration: We aim to discover synonyms for a given query term from privacy-aware clinical data by effectively leveraging two important types of information: Surface form and global contexts.

For the sake of patient privacy and security, it is usually quite difficult, if not impossible, for medical institutes to grant public access to large-scale raw or even de-identified clinical texts [2]. Consequently, medical terms[1] and their aggregated co-occurrence counts extracted from raw clinical texts are becoming a popular (although not perfect) substitute for raw clinical texts for the research community to study EMR data [2, 8, 33]. For example, Finlayson et al. [8] released millions of medical terms extracted from the clinical texts in Stanford Hospitals and Clinics as well as their global co-occurrence counts, rather than releasing raw sentences/paragraphs/documents from the clinical text corpus. In this work, we refer to the given set of medical terms and their co-occurrence statistics in a clinical text corpus as *privacy-aware* clinical data, and investigate synonym discovery task on such data (Figure 1): *Given a set of terms extracted from clinical texts as well as their global co-occurrence graph*[2]*, recommend a list of synonyms for a query term.*

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330894

[1] A medical term is a single- or multi-word string (e.g., "Aspirin", "Acetylsalicylic Acid").
[2] where each node is a medical term and each edge between two nodes is weighted by the number of times that two terms co-occur in a given context window.

Developing effective approaches under this setting is particularly meaningful, as they will suggest that one can utilize less sensitive information (i.e., co-occurrence statistics rather than raw sentences in clinical texts) to perform the task well.

A straightforward approach to obtain synonyms is to map the query term to a knowledge base (KB) entity and retrieve its synonyms or aliases stored in the KBs. However, it is widely known that KBs are incomplete and outdated, and their coverage of synonyms can be very limited [38]. In addition, the informal writing of clinical texts often contain variants of surface forms, layman terms, frequently misspelling words, and locally practiced abbreviations, which should be mined to enrich synonyms in KBs. Recent works [30, 37, 42] have been focused on automatic synonym discovery from massive text corpora such as Wikipedia articles and PubMed paper abstracts. When predicting if two terms are synonyms or not, such approaches usually leverage the original sentences (a.k.a. *local* contexts) mentioning them, and hence do not apply or work well under our privacy-aware data setting where such sentences are unavailable.

Despite the lack of local contexts, we observe two important types of information carried in the privacy-aware data - surface form information and global context information (i.e., co-occurrence statistics). In this work, we aim to effectively leverage these two types of information for synonym discovery, as shown in Figure 1.

Some recent works [24, 25] model the similarity between terms in the character-level. For example, Mueller and Thyagarajan [24] learn the similarity between two sequences of characters, which can be applied for discovering synonyms that look alike such as "vit c" and "vitamin c". However, we observe two common phenomena that such approaches cannot address well and would induce false positive and false negative predictions respectively: (1) Some terms are similar in surface form but do not have the same meaning (e.g., "hemostatic" and "homeostasis", where the former means a process stopping bleeding while the latter refers to a constant internal environment in the human body); (2) Some terms have the same meaning but are different in surface form (e.g., "ascorbic acid" and "vitamin c" are the same medicinal product but look different).

On the other hand, given a term co-occurrence graph, various distributional embedding methods such as [18, 28, 34] have been proposed to learn a distributional representation (a.k.a. embedding) for each term based on its *global* contexts (i.e., terms connected to it in the co-occurrence graph). The main idea behind such methods is that two terms should have similar embedding vectors if they share a lot of global contexts. However, we observe that the privacy-aware clinical data tends to be very *noisy* due to the original data processing procedure[3], which presents new challenges for utilizing global contexts to model semantic similarity between terms. For example, Finlayson et al. [8] prune the edges between two terms co-occurring less than 100 times, which can lead to missing edges between two related terms in the co-occurrence graph. Ta et al. [33] remove all concepts with singleton frequency counts below 10. Hence, the noisy nature of the co-occurrence graph makes it less accurate to embed a term based on their original contexts. Moreover, when performing the synonym discovery task, users are very likely to issue a query term that does not appear in the given co-occurrence data. We refer to such query terms as Out-of-Vocabulary (OOV). Unlike In-Vocabulary[4] query terms, OOV query terms do not have their global contexts readily available in the given graph, which makes synonym discovery even more challenging.

In this paper, to address the above challenges and effectively utilize both the surface form and the global context information in the privacy-aware clinical data, we propose a novel framework named SurfCon which consists of a bi-level surface form encoding component and a context matching component, both based on neural models. The bi-level surface form encoding component exploits both character- and word-level information to encode a medical term into a vector. It enables us to compute a surface score of two terms based on their encoding vectors. As mentioned earlier, such surface score works well for detecting synonyms that look similar in surface form. However, it tends to miss synonymous terms that do not look alike. Therefore, we propose the context matching component to model the semantic similarity between terms, which plays a complementary role in synonymy discovery.

Our context matching component first utilizes the bi-level surface form encoding vector for a term to predict its potential global contexts. Using predicted contexts rather than the raw contexts in the given graph enables us to handle OOV query terms and also turns out to be effective for InV query terms. Then we generate a semantic vector for each term by aggregating the semantic features from predicted contexts using two mechanisms - static and dynamic representation mechanism. Specifically, given term a and term b, the dynamic mechanism aims to learn to weigh the importance of individual terms in a's contexts based on their semantic matching degree with b's contexts, while the static mechanism assigns equal weights to all terms in one's contexts. The former takes better advantage of individual terms within the contexts and empirically demonstrates superior performance.

Our contributions are summarized in three folds:

- We study the task of synonym discovery under a new setting, i.e., on privacy-aware clinical data, where only a set of medical terms and their co-occurrence statistics are given, and local contexts (e.g., sentences mentioning a term in a corpus) are not available. It is a practical setting given the wide concern about patient privacy for access to clinical texts and also presents unique challenges to address for effective synonym discovery.
- We propose a novel and effective framework named SurfCon that can discover synonyms for both In-Vocabulary (InV) and Out-of-Vocabulary (OOV) query terms. SurfCon considers two complementary types of information based on neural models - surface form information and global context information of a term, where the former works well for detecting synonyms that are similar in surface form while the latter can help better find synonyms that do not look alike but are semantically similar.
- We conduct extensive experiments on publicly available privacy-aware clinical data and demonstrate the effectiveness of our framework in comparison with various baselines and our own model variants.

[3]This tends to be a common issue in many scenarios as raw data has to go through various pre-processing steps for privacy concerns.

[4]Query terms that appear in the given co-occurrence graph are referred to as In-Vocabulary (InV).

2 TASK SETTING

In this section, we clarify several terminologies used in this paper as well as our problem definition:

Privacy-aware Clinical Data. Electronic medical records (EMRs) typically contain patient medical information such as discharge summary, treatment, and medical history. In EMRs, a significant amount of clinical information remains under-tapped in the unstructured clinical texts. However, due to privacy concerns, access to raw or even de-identified clinical texts in large quantities is quite limited. Also, traditional de-identification methods, e.g., removing the 18 HIPAA identifiers [32], require significant manual efforts for the annotation [7]. Moreover, there also exists the risk that de-identified data can be attacked and recovered by the re-identification in some cases [9]. Thus, to facilitate research on EMRs, an increasingly popular substitute strategy for releasing raw clinical texts is to extract medical terms and their aggregated co-occurrence counts from the corpus [2, 8, 33]. We refer to such data as privacy-aware clinical data in this paper. Converting raw sentences to co-occurrence data protects privacy as original patient records are very unlikely to be recovered. However, the local context information contained in the raw sentences is also lost, which makes various tasks including synonym discovery more challenging under privacy-aware datasets.

Medical Term Co-occurrence Graph. A medical term-term co-occurrence graph is defined as G=(V, E), where V is the set of vertices, each representing a medical term extracted from clinical texts. Each vertex has a surface form string (e.g., "vitamin c", "cancer") which is the spelling of the medical term. E is the set of edges, each weighted by how many times two terms co-occur in a certain context window (e.g., notes from patient records within 1 day).

Medical Term Synonym. Synonyms of a medical term refer to other medical terms that can be used as its alternative names [30]. For example, "vit c", "c vitamin" and "ascorbic acid" refer to the same medicinal product, while "Alzheimer's disease" and "senile dementia" represent the same disease. In our dataset, the extracted medical terms are mapped to the Unified Medical Language System (UMLS) [3] Concept Unique Identifier (CUI) by [8]. Different terms mapping to the same UMLS CUI are treated as synonyms for model training/development/testing.

Task Definition. We formally define our task of synonym discovery on privacy-aware clinical data as: *Given a medical term co-occurrence graph G, for a query term q (which can be either In-Vocabulary or Out-of-Vocabulary), recommend a list of medical terms from G that are likely to be synonyms of q.*

3 SURFCON FRAMEWORK

In this section, we introduce our proposed framework SurfCon for synonym discovery on privacy-aware clinical data.

3.1 Overview

We observe two important types of information carried in the privacy-aware clinical data: surface form information of a medical term and the global contexts from the given co-occurrence graph. On the one hand, existing approaches [25] using character-level features to detect synonyms could work well when synonyms share a high string similarity, but tend to produce false positive predictions (when two terms look similar but are not synonyms, e.g., "hemostatic" and "homeostasis") and false negative predictions (when two terms are synonyms but look very different, e.g., "ascorbic acid" and "vitamin c"). On the other hand, the global contexts of a term under the privacy-aware setting tend to be noisy partly due to the original data pre-processing procedure, which also presents challenges for using them to model the semantic similarity between terms. Thus, a framework that is able to effectively leverage these two types of information needs to be carefully designed.

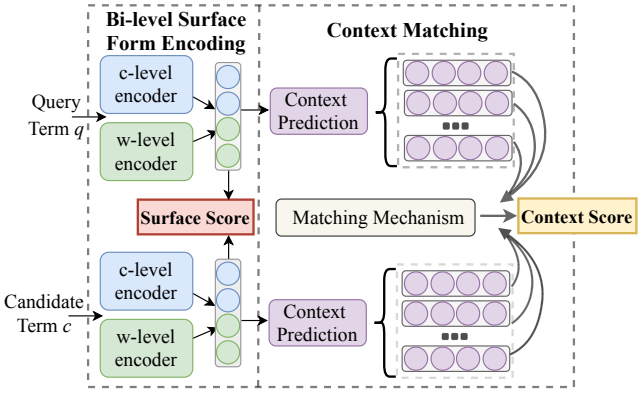


Figure 2: Framework overview. For each query term, a list of candidate terms will be ranked based on both the surface and context scores.

Towards that end, we propose SurfCon (Figure 2) and summarize its high-level ideas as below:

(1) Given a query term (whether being InV or OOV), the bi-level surface form encoding component and the context matching component score a candidate term[5] respectively based on the surface form information and global context information. The former enables us to find synonyms that look similar to the query term by considering both character- and word-level information, and the latter complements it by capturing the semantic similarity between terms to better address the false positive and false negative problem mentioned earlier.

(2) Considering the original global contexts being noisy as well as the existence of OOV query terms, instead of directly leveraging the raw global contexts, the context matching component will first utilize the surface form encoding vector of a term to *predict* its potential global contexts[6]. We then investigate a novel dynamic context matching mechanism (see Section 3.2.2 for details) to evaluate if two terms are synonyms based on their predicted contexts.

(3) The two components are combined by a weighted score function, in which parameters are jointly optimized with a widely used ranking algorithm ListNet [5]. At testing time, given a query term, candidate terms are ranked based on the optimized score function.

3.2 Methodology

Now we describe the two components of SurfCon: Bi-level Surface Form Encoding and Context Matching in details.

3.2.1 ***Bi-level Surface Form Encoding.*** The bi-level surface form encoding of our framework aims to model the similarity between two terms at the surface form level, as we observe that two terms

[5]Every term in the given co-occurrence graph can be a candidate term.

[6]For terms in the co-occurrence graph, predicting contexts can be treated as denoising its original global contexts (or edges)

tend to be synonymous if they are very similar in surface forms. Such observation is intuitive but works surprisingly well in synonym discovery task. Driven by this observation, we design the bi-level surface form encoding component in a way that both of character- and word-level information of terms are captured. Then, a score function is defined to measure the surface form similarity for a pair of terms based on their bi-level encoding vectors. The bi-level encoders are able to encode surface form information of both InV terms and OOV terms.

Specifically, as shown in Figure 2, given a query term q and a candidate term c, we denote their character-level sequences as $x_q = \{x_{q,1}, ..., x_{q,m_q}\}, x_c = \{x_{c,1}, ..., x_{c,m_c}\}$, and their word-level sequences as $w_q = \{w_{q,1}, ..., w_{q,n_q}\}, w_c = \{w_{c,1}, ..., w_{c,n_c}\}$, where m_q, n_q, m_c, n_c are the length of the character-level sequence and word-level sequence of the query term and the candidate term respectively. Then we build two encoders ENC^{ch} and ENC^{wd} to capture the surface form information at the character- and word-level respectively:

$$\begin{aligned} s_q^{ch} &= \text{ENC}^{ch}(x_{q,1}, \ldots, x_{q,m_q}),\ s_q^{wd} = \text{ENC}^{wd}(w_{q,1}, \ldots, w_{q,n_q}) \\ s_c^{ch} &= \text{ENC}^{ch}(x_{c,1}, \ldots, x_{c,m_c}),\ s_c^{wd} = \text{ENC}^{wd}(w_{c,1}, \ldots, w_{c,n_c}) \end{aligned} \tag{1}$$

where $s_q^{ch}, s_c^{ch} \in \mathbb{R}^{d_c}$ are the character-level embeddings for the query and candidate terms, and $s_q^{wd}, s_c^{wd} \in \mathbb{R}^{d_w}$ are the word-level embeddings for the query and candidate terms respectively.

Note that there has been a surge of effective encoders that model sequential information from character-level or word-level, ranging from simple look-up table (e.g., character n-gram [13] and Skip-Gram [23]) to complicated neural network architectures (e.g., CNN [14], LSTM [1] and Transformer [35], etc.). For simplicity, here, we adopt simple look-up tables for both character-level embeddings and word-level embeddings. Instead of randomly initializing them, we borrow pre-trained character n-gram embeddings from Hashimoto et al. [13] and word embeddings from Pennington et al. [28]. Our experiments also demonstrate that these simple encoders can well encode surface form information of medical terms for synonym discovery task. We leave evaluating more complicated encoders as our future work.

After we obtain the embeddings at both levels, we concatenate them and apply a nonlinear function to get the surface vector s for the query and candidate term. Let us denote such encoding process as a function $h(\cdot)$ with the input as term q or c and the output as the surface vector s_q or s_c:

$$\begin{aligned} s_q &= h(q) = \tanh([s_q^{ch}, s_q^{wd}]W_s + b_s), \\ s_c &= h(c) = \tanh([s_c^{ch}, s_c^{wd}]W_s + b_s) \end{aligned} \tag{2}$$

where the surface vectors $s_q, s_c \in \mathbb{R}^{d_s}$, and $W_s \in \mathbb{R}^{(d_c+d_w)\times d_s}, b_s \in \mathbb{R}^{d_s}$ are weight matrix and bias for a fully-connected layer.

Next, we define the surface score for a query term q and a candidate term c to measure the surface form similarity based on their encoding vectors s_q and s_c:

$$\text{Surface Score}\,(q, c) = f_s(s_q, s_c) \tag{3}$$

3.2.2 ***Context Matching.*** In order to discover synonyms that are not similar in surface form, and also observing that two terms tend to be synonyms if their global contexts in the co-occurrence graph are semantically very relevant, we design the context matching component to capture the semantic similarity of two terms by carefully leveraging their global contexts. We first illustrate the intuition behind this component using a toy example:

Example 1. **[Toy Example for Illustration.]** *Assume we have a query term "vitamin c" and a candidate term "ascorbic acid". The former is connected with two terms "iron absorption" and "vitamin b" in the co-occurrence graph as global contexts, while the latter has "fatty acids" and "anemia" as global contexts.*

Our context matching component essentially aims to use a term's contexts to represent its semantic meaning and a novel *dynamic context matching mechanism* is developed to determine the importance of each individual term in one's contexts. For example, *"iron absorption"* is closely related to *"anemia"* since the disease "anemia" is most likely to be caused by the iron deficiency. Based on the observation, we aim to increase the relative importance of *"iron absorption"* and *"anemia"* in their respective context sets when representing the semantic meaning of *"vitamin c"* and *"ascorbic acid"*. Therefore, we develop a novel dynamic context matching mechanism to be introduced shortly.

In order to recover global contexts for OOV terms and also noticing the noisy nature of the co-occurrence graph mentioned earlier, we propose an *inductive context prediction module* to predict the global contexts for a term based on its surface form information instead of relying on the raw global contexts in the given co-occurrence graph.

Inductive Context Prediction Module. Let us first denote a general medical term as t. For a term-term co-occurrence graph, we treat all InV terms as possible context terms and denote them as $\{u_j\}_{j=1}^{|V|}$ where $|V|$ is the total number of terms in the graph. The inductive context prediction module aims to predict how likely term u_j appears in the context of t (denoted as the conditional probability $p(u_j|t)$). To learn a good context predictor, we utilize all existing terms in the graph as term t, i.e., $t \in \{u_i\}_{i=1}^{|V|}$ and the conditional probability becomes $p(u_j|u_i)$.

Formally, the probability of observing term u_j in the context of term u_i is denoted as:

$$p(u_j|u_i) = \frac{\exp(v_{u_j}^T \cdot s_{u_i})}{\sum_{k=1}^{|V|} \exp(v_{u_k}^T \cdot s_{u_i})} \tag{4}$$

where $s_{u_i} = h(u_i)$ and $h(\cdot)$ is the same encoder function defined in section 3.2.1. $v_{u_j} \in \mathbb{R}^{d_o}$ is the context embedding vector corresponding to term u_j and we let $d_o = d_s$. The predicted distribution $p(u_j|u_i)$ is optimized to be close to the empirical distribution $\hat{p}(u_j|u_i)$ defined as:

$$\hat{p}(u_j|u_i) = \frac{w_{ij}}{\sum_{(i,k)\in E} w_{ik}} \tag{5}$$

where E is the set of edges in the co-occurrence graph and w_{ij} is the weight between term u_i and term u_j. We adopt the cross entropy loss function for optimizing:

$$L_n = -\sum_{u_i, u_j \in V} \hat{p}(u_j|u_i) \log(p(u_j|u_i)) \tag{6}$$

When the number of terms in the graph $|V|$ is very large, it is computationally costly to calculate the conditional probability $p(u_j|u_i)$, and one can utilize the negative sampling algorithm [22]

to train our inductive context predictor efficiently. The loss function Eqn. 6 can be modified as:

$$\log \sigma(v_{u_j}^T \cdot s_{u_i}) + \sum_{n=1}^{N_0} E_{u_n \sim P_n(u)}[\log \sigma(-v_{u_n}^T \cdot s_{u_i})] \tag{7}$$

where $\sigma(x) = 1/(1+\exp(-x))$ and u_n is the negative sample drawn from the noise distribution $P_n(u) \propto d_u^{3/4}$. N_0 is the number of negative samples and d_u is the degree of term u in the co-occurrence graph.

Now, given a term t (either InV or OOV), we can select the top-K terms as its predicted contexts based on the predicted probability distribution $p(\cdot|t)$. Next, we describe the dynamic context matching mechanism to model the semantic similarity of two terms based on their predicted contexts.

Dynamic Context Matching Mechanism. Inspired by previous works on neighborhood aggregation based graph embedding methods [12, 36], which generate an embedding vector for an InV node by aggregating features from its neighborhood (contexts), we introduce two semantic vectors respectively for the query term and the candidate term, $v_q, v_c \in \mathbb{R}^{d_e}$, and learn them by aggregating the feature vectors of their corresponding top-K predicted contexts from previous module.

Let us define $v_q^i \in \mathbb{R}^{d_e}$ as the feature vector of the i-th term in query term q's context while $v_c^j \in \mathbb{R}^{d_e}$ as the feature vector of the j-th term in candidate term c's context, and their context sets as $\Phi(q) = \{v_q^i\}_{i=1}^K$, $\Phi(c) = \{v_c^j\}_{j=1}^K$. Essentially, as we aim to capture the semantic meaning of terms, the feature vectors v_q^i's and v_c^j's are expected to contain semantic information. Also noticing that all predicted context terms are InV terms (i.e., in the co-occurrence graph), which allows us to adopt widely used graph embeddings, such as LINE(2nd) [34] as their feature vectors.

One naive way to obtain the context semantic vectors, v_q and v_s is to average vectors in their respective context set. Since such v_q (or v_c) does not depend on the other one, we refer to such vectors as "static" representations for terms.

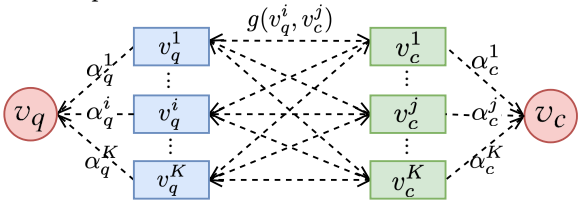


Figure 3: Dynamic Context Matching Mechanism.

In contrast to the static approach, we propose the *dynamic context matching mechanism* (as shown in Figure 3), which weighs each term in the context of q (or c) based on its matching degree with terms in the context of c (or q) and hence the context semantic vector representation v_q (or v_c) is *dynamically* changing depending on which terms it is comparing with. More specifically, let us define $g(x, y) = \tanh(xW_m y^T)$ as a nonlinear function parameterized with weight matrix $W_m \in \mathbb{R}^{d_e \times d_e}$ to measure the similarity between two row vectors x and y. For each context vector v_q^i of the query term, we calculate its weight based on how it matches with c's contexts overall:

$$\text{match}[v_q^i, \Phi(c)] = \text{Pooling}[g(v_q^i, v_c^1), ..., g(v_q^i, v_c^K)] \tag{8}$$

For the pooling operation, we empirically choose the mean pooling strategy as it performs better than alternatives such as max pooling in our experiments. Then we normalize the weight of v_q^i as:

$$\alpha_q^i = \frac{e^{\text{match}[v_q^i, \Phi(c)]}}{\sum_{k=1}^K e^{\text{match}[v_q^k, \Phi(c)]}} \tag{9}$$

Finally, the context semantic vector for the query term v_q is calculated through a weighted combination of q's contexts:

$$v_q = \sum_{i=1}^K \alpha_q^i \cdot v_q^i \tag{10}$$

Following the same procedure, we can obtain the context semantic vector v_c for the candidate term w.r.t. the query term. Then we define the context score for a query term q and a candidate term c to measure their semantic similarity based on v_q and v_c:

$$\text{Context Score}(q, c) = f_c(v_q, v_c) \tag{11}$$

3.3 Model Optimization and Inference

Objective Function. Given a query term q and a candidate term c, to capture their similarity based on surface forms and global contexts, we define the final score function as:

$$f(q, c) = (1-\gamma) \cdot f_s(s_q, s_c) + \gamma \cdot f_c(v_q, v_c) \tag{12}$$

$f_s(\cdot)$ and $f_c(\cdot)$ are similarity functions between two vectors, e.g., cosine similarity or bilinear similarity. Now we obtain the recommendation probability of each candidate $t_i \in \{t_1, ..., t_N\}$ given a query q:

$$p(t_i|q) = \frac{e^{f(q,t_i)}}{\sum_{k=1}^N e^{f(q,t_k)}} \tag{13}$$

where N is the size of the candidate set. Finally, we adopt the ListNet [5] ranking framework which minimizes the cross entropy loss for query term q:

$$L_r = -\sum_{i=1}^N p^*(t_i|q) \log p(t_i|q) \tag{14}$$

where $p^*(t_i|q)$ is the normalized ground-truth distribution of a list of ranking scores as $\{r_i\}_{i=1}^N$ where r_i equals to 1 if q and t_i are synonyms and 0 otherwise.

Training. For efficiency concerns, we adopt a two-phase training strategy: We first train the inductive context prediction module by loss function L_n (Eqn. 6) in the term-term co-occurrence graph, and sample top-K contexts based on the predicted probability distribution and use them in the context matching component. Then, we train the ranking framework by minimizing the ranking loss L_r (Eqn. 14).

Inference. At the inference stage, we treat all InV terms as candidates for a given query. Since the dynamic representation mechanism involves pairwise term matching between the contexts of the query term and those of each candidate term and can have a high computational cost when the candidate set size is large, we adopt a two-step strategy: (1) For a given query term, select its top-N high potential candidates based on the surface form encoding vector and the context semantic vector obtained by the static representation mechanism; (2) Re-rank the selected candidates by applying our SurfCon framework with the dynamic representation mechanism.

4 EXPERIMENTS

Now we evaluate our proposed framework SurfCon to show the effectiveness of leveraging both surface form information and global context information for synonym discovery.

4.1 Datasets

Medical Term Co-occurence Graph. We adopt publicly available sets of medical terms with their co-occurrence statistics which are extracted by Finlayson et al. [8] from 20 million clinical notes collected from Stanford Hospitals and Clinics[20] since 1995. Medical terms are extracted using an existing phrase mining tool [16] by matching with 22 clinically relevant ontologies such as SNOMED-CT and MedDRA. And co-occurrence frequencies are counted based on how many times two terms co-occur in the same temporal *bin* (i.e., a certain timeframe in patient's records), e.g., 1, 7, 30, 90, 180, 365, and ∞-day *bins*.

Without loss of generality, we choose 1-day per-bin and ∞-day per-bin[7] graphs to evaluate different methods. We first convert the global counts between nodes to the PPMI values [17] and adopt subsampling [23] to filter very common terms, such as "medical history", "medication dose", etc. We choose these two datasets because they have very different connection density as shown in Table 1, and denote them as **1-day** and **All-day** datasets.

Synonym Label. In the released datasets, Finlayson et al. [8] provided a term-to-UMLS CUI mapping based on the same 22 ontologies as used when extracting terms. They reduced the ambiguity of a term by suppressing its least likely meaning so as to provide a high-quality mapping. We utilized such mapping to obtain the synonym labels: Terms mapped to the same UMLS CUI are treated as synonyms, e.g., terms like "c vitamin", "vit c", "ascorbic acid" are synonyms as they are all mapped to the concept "Ascorbic Acid" with ID C0003968.

Query Terms. Given a medical term-term co-occurrence graph, terms in the graph that can be mapped to UMLS CUIs are treated as potential query terms, and we split all such terms into training, development and testing sets. Here, since all terms appear in the given co-occurrence graph, this testing set is referred to as the **InV testing set**. We also create an **OOV testing set**: Under a UMLS CUI, terms not in the co-occurrence graph are treated as OOV query terms and are paired with their synonyms which are in the graph to form positive pairs. We sample 2,000 of such OOV query terms for experiments. In addition, since synonyms with different surface forms tend to be more challenging to discover (e.g., "vitamin c" vs. "ascorbic acid"), we also sample a subset named **Dissim** under both InV and OOV testing set, where query terms paired with their dissimilar synonyms[8] are selected. Statistics of our training/dev/testing sets are given in Table 1.

4.2 Experimental Setup

4.2.1 Baseline methods. We compare SurfCon with the following 10 methods. The baselines can be categorized by three types: (i) Surface form based methods, which focus on capturing the surface form information of terms. (ii) Global context based methods, which try to learn embeddings of terms for synonym discovery; (iii) Hybrid methods, which combine surface form and global context information. The others are our model variants.

Table 1: Statistics of our datasets.

		1-day dataset	All-day dataset
# Nodes		52,804	43,406
# Edges		16,197,319	50,134,332
Average # Degrees		613.5	2310.0
# Train Terms		9,451	7,021
# Dev Terms		960	726
# InV Test Terms	All	960	726
	Dissim	175	152
# OOV Test Terms	All	2,000	2,000
	Dissim	809	841

Surface form based methods. (1) *CharNgram* [13]: We borrow pre-trained character n-gram embeddings from Hashimoto et al. [13] and take the average of unique n-gram embeddings for each term as its feature, and then train a bilinear scoring function following previous works [30, 42]. (2) *CHARAGRAM* [40]: Similar as above, but we further fine-tune CharNgram embeddings using synonym supervision. (3) *SRN* [25]: A Siamese network structure is adopted with a bi-directional LSTM to encode character sequence of each term and cosine similarity is used as the scoring function.

Global context based methods. (4) *Word2vec* [23]: A popular distributional embedding method. We obtain word2vec embeddings by doing SVD decomposition over the Shifted PPMI co-occurrence matrix [18]. We treat the embeddings as features and use a bilinear score function for synonym discovery. (5) *LINE(2nd)* [34]: A widely-adopted graph embedding approach. Similarly, embeddings are treated as features and a bilinear score function is trained to detect synonyms. (6) *DPE-NoP* [30]: DPE is proposed for synonym discovery on text corpus, and consists of a distributional module and a pattern module, where the former utilizes global context information and the latter learns patterns from raw sentences. Since raw texts are unavailable in our setting, we only deploy the distributional module (a.k.a. DPE-NoP in Qu et al. [30]).

Hybrid methods. (7) *Concept Space Model* [37]: A medical synonym extraction method that combines word embeddings and heuristic rule-based string features. (8) *Planetoid* [41]: An inductive graph embedding method that can generate embeddings for both observed and unseen nodes. We use the bi-level surface form encoding vectors as the input and take the intermediate hidden layer as embeddings. Similarly, a bilinear score function is used for synonym discovery.

Model variants. (9) *SurfCon (Surf-Only)*: A variant of our framework which only uses the surface score for ranking. (10) *SurfCon (Static)*: Our framework with static representation mechanism. By comparing these variants, we verify the performance gain brought by modeling global contexts using different matching mechanisms.

For baseline methods (1-3 and 8) and our models, we test them under both InV and OOV settings. For the others (4-7), because they rely on embeddings that are only available for InV terms, we only test them under InV setting.

4.2.2 Candidate Selection and Performance Evaluation. For evaluating baseline methods and our model, we experiment with two strategies: (1) Random candidate selection. For each query term, we randomly sample 100 non-synonyms as negative samples and mix

[7] Per-bin means each unique co-occurring term-term pair is counted at most once for each relevant bin of a patient. We refer readers to Finlayson et al. [8] for more information.

[8] Dissimilarity is measured by Levenshtein edit distance [10] with a threshold (0.8).

Table 2: Model evaluation in MAP with random candidate selection.

Method Category	Methods	**1-day** Dataset					**All-day** Dataset				
		Dev	InV Test		OOV Test		Dev	InV Test		OOV Test	
			All	Dissim	All	Dissim		All	Dissim	All	Dissim
Surface form based methods	CharNgram [13]	0.8755	0.8473	0.4657	0.7427	0.4131	0.8652	0.8553	0.4615	0.7675	0.4424
	CHARAGRAM [40]	0.8705	0.8507	0.5504	0.7609	0.5142	0.8915	0.8805	0.5153	0.8119	0.5282
	SRN [25]	0.8886	0.8565	0.5102	0.7241	0.4341	0.8460	0.8170	0.4523	0.7110	0.4176
Global context based methods	Word2vec [23]	0.3838	0.3748	0.3188	-	-	0.4801	0.476	0.4180	-	-
	LINE(2nd) [34]	0.4279	0.4301	0.3494	-	-	0.5068	0.5043	0.4369	-	-
	DPE-NoP [30]	0.6222	0.6107	0.4855	-	-	0.5928	0.5949	0.4938	-	-
Hybrid methods (surface+context)	Concept Space [37]	0.8094	0.8109	0.4690	-	-	0.8064	0.7924	0.5574	-	-
	Planetoid [41]	0.8813	0.8514	0.5612	0.731	0.4714	0.8818	0.8765	0.6963	0.7403	0.4986
Our model and variants	SurfCon (Surf-Only)	0.9160	0.9053	0.6145	0.8228	0.5829	0.9034	0.8958	0.6006	0.8183	0.5622
	SurfCon (Static)	0.9242	0.9151	0.6542	0.8285	0.5933	0.9170	0.9019	0.6656	0.8203	0.5664
	SurfCon	**0.9348**	**0.9176**	**0.6821**	**0.8301**	**0.6009**	**0.9219**	**0.9199**	**0.7171**	**0.8232**	**0.5673**

them with synonyms for testing. This strategy is widely adopted by previous work on synonym discovery for testing efficiency [37, 42]. (2) Inference-stage candidate selection. As mentioned in section 3.3, at the inference stage, we first obtain high potential candidates in a lightweight way. Specifically, after the context predictor is pre-trained, for all terms in the given graph as well as the query term, we generate their surface form vector s and context semantic vector v obtained by the static representation. Then we find top 50 nearest neighbors of the query term respectively based on s and v using cosine similarity. Finally, we apply our methods and baselines to re-rank the 100 high potential candidates. We refer to these two strategies as *random candidate selection* and *inference-stage candidate selection*.

For evaluation, we adopt a popular ranking metric Mean Average Precision defined as $\text{MAP} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{m_i}\sum_{j=1}^{m_i}\text{Precision}(R_{ij})$, where R_{ij} is the set of ranked terms from 1 to j, m_i is the length of i-th list, and $|Q|$ is the number of queries.

4.2.3 Implementation details. Our framework is implemented in Pytorch [27] with Adam optimizer [15]. The dimensions of character embeddings (d_c), word embeddings (d_w), surface vectors (d_s), and sementic vectors (d_e) are set to be 100, 100, 128, 128. Early stopping is used when the performance in the dev sets does not increase continuously for 10 epochs. We directly optimize Eqn. 6 since the number of terms in our corpus is not very large, and set $f_s(\cdot)$ and $f_c(\cdot)$ to be cosine similarity and bilinear similarity function respectively, based on the model performance on the dev sets. When needed, string similarities are calculated by using the Distance package[9]. Pre-trained CharNgram [13] embeddings are borrowed from the authors[10]. For CHARAGRAM [40], we initialize the n-gram embeddings by using pre-trained CharNgram and fine-tune them on our dataset by the synonym supervision. We learn LINE(2nd) embeddings [34] by using OpenNE[11]. Heuristic rule-based matching features of Concept Space model are implemented according to [37]. Code, datasets, and more implementation details are available online[12].

[9]https://github.com/doukremt/distance
[10]https://github.com/hassyGo/charNgram2vec
[11]https://github.com/thunlp/OpenNE
[12]https://github.com/yzabc007/SurfCon

4.3 Results and Analysis

4.3.1 Evaluation with Random Candidate Selection. We compare all methods under random candidate selection strategy with the results shown in Table 2.

(1) Comparing SurfCon with surface form based methods. Our model beats all surface form based methods, including strong baselines such as SRN that use complicated sequence models to capture character-level information. This is because: 1) Bi-level encoder of SurfCon could capture surface form information from both character- and word-level, while baselines only consider either of them; 2) SurfCon captures global context information, which could complement surface form information for synonym discovery. In addition, in comparison with CharNgram and CHARAGRAM, our model variant SurfCon (Surf-Only), which also only uses surface form information, obtains consistently better performance, especially in the OOV Test set. The results demonstrate that adding word-level surface form information is useful to discover synonyms.

(2) Comparing SurfCon with global context based methods. SurfCon substantially outperforms all other global context based methods (Word2vec, LINE(2nd) and DPE-NoP). This is largely due to the usage of surface form information. In fact, as one can see, global context based methods are generally inferior to surface form based methods, partly due to the fact that a large part of synonyms are similar in surface form, while only a small portion of them are in very different surface form. Thus, detecting synonyms without leveraging surface information can hardly lead to good results. Besides, our context matching component conducts context prediction and matching strategies, which takes better advantage of global context information and thus lead to better performance on the synonym discovery task.

(3) Comparing SurfCon with hybrid methods. We also compare our model with baselines that combine both surface form and global context information. First, SurfCon is superior to the concept space model because the latter simply concatenates distributional embeddings with rule-based string features, e.g., the number of shared words as features and apply a logistic regression classifier for classification. Further, SurfCon also performs better than Planetoid, partly because our framework more explicitly leverages both surface form and global context information to formulate

Table 3: Model evaluation at inference stage.

Methods	1-day		All-day	
	InV Test	OOV Test	InV Test	OOV Test
CHARAGRAM [13]	0.3921	0.4044	0.3941	0.3913
DPE-NoP [30]	0.2396	-	0.2408	-
Planetoid [41]	0.4563	0.4268	0.3765	0.3812
SurfCon	0.5525	0.5068	0.4686	0.4661

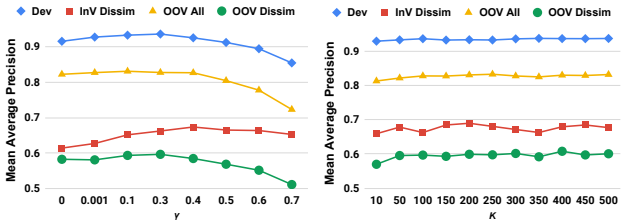


Figure 4: Performance w.r.t. (a) the coefficient of context score γ and (b) the number of context terms K.

synonym scores, while Planetoid relies on one embedding vector for each term which only uses surface form information as input.

(4) Comparing SurfCon with its variants. To better understand why SurfCon works well, we compare it with several variants. Under both datasets, SurfCon (Surf-Only) already outperforms all baselines demonstrating the effectiveness of our bi-level surface form encoding component. With the context matching component in SurfCon (Static), the performance is further improved, especially under *InV Test Dissim* setting where synonyms tend to have different surface forms and we observe around 4% performance gain. Further, by using dynamic representation in context matching mechanism, SurfCon obtains better results, which demonstrates that the dynamic representation is more effective to utilize context information compared with the static strategy.

4.3.2 Evaluation at Inference Stage. To further evaluate the power of our model in real practice, we test its performance at the inference stage as mentioned in section 3.3. Due to space constraint, we only show the comparison in Table 3 between SurfCon and several strong baselines revealed by Table 2. In general, the performance of all methods decreases at the inference stage compared with the random candidate selection setting, because the constructed list of candidates becomes harder to rank since surface form and context information are already used for the construction. For example, a lot of non-synonyms with similar surface form are often included in the candidate list. Even though the task becomes harder, we still observe our model outperforms the strong baselines by a large margin (e.g., around 8% at least) under all settings.

4.3.3 Parameter Sensitivity. Here we investigate the effect of two important hyper-parameters: The coefficient γ which balances the surface score and the context score, and the number of predicted contexts K used for context matching. As shown in Figure 4(a), the performance of SurfCon first is improved as γ increases, which is expected because as more semantic information is incorporated, SurfCon could detect more synonyms that are semantically similar. When we continue to increase γ, the performance begins to decrease and the reason is that surface form is also an important source of information that needs to be considered. SurfCon achieves the best performance roughly at $\gamma = 0.3$ indicating surface form information is relatively more helpful for the task than global context information. This also aligns well with our observation that synonyms more often than not have similar surface forms. Next, we show the impact of K in Figure 4(b). In general, when K is small (e.g., $K = 10$), the performance is not as good since little global context information is considered. Once K increases to be large enough (e.g., ≥ 50), the performance is not sensitive to the variation under most settings showing that we can choose smaller K for computation efficiency but still with good performance.

Table 4: Case studies on the 1-day dataset. Bold terms are synonyms in our labeled set while underlined terms are not but quite similar to the query term in semantics.

Query Term	"unable to vocalize" (InV)	"marijuana" (OOV)
SurfCon Top Ranked Candidates	"does not vocalize" "aphonia" "loss of voice" "vocalization" **"unable to phonate"**	**"marijuana abuse"** **"cannabis"** "cannabis use" "marijuana smoking" "narcotic"
Labeled Synonym Set	"unable to phonate"	"cannabis" "marijuana abuse" "marihuana abuse"

4.4 Case Studies

We further conduct case studies to show the effectiveness of SurfCon. Two query terms "unable to vocalize" and "marijuana" are chosen respectively from the InV and OOV test set where the former is defined as the inability to produce voiced sound and the latter is a psychoactive drug used for medical or recreational purposes. As shown in Table 4, for the InV query "unable to vocalize", our model can successfully detect its synonyms such as "unable to phonate", which already exists in the labeled synonym set collected based on term-to-UMLS CUI mapping as we discussed in Section 2. More impressively, our framework also discovers some highly semantically similar terms such as "does not vocalize" and "aphonia", even if some of them are quite different in surface form from the query term. For the OOV query "marijuana", SurfCon ranks its synonym "marijuana abuse" and "cannabis" at a higher place. Note that the other top-ranked terms are also very relevant to "marijuana".

5 RELATED WORK

Character Sequence Encoding. To capture the character-level information of terms, neural network models such as Recurrent Neural Networks and Convolutional Neural Networks can be applied on character sequences [1, 14]. Further, CHARAGRAM [40], FastText [4], and CharNGram [13] are proposed to represent terms and their morphological variants by capturing the shared subwords and n-grams information. However, modeling character-level sequence information only is less capable of discovering semantically similar synonyms, and our framework considers global context information to discover those synonyms.

Word and Graph/Network Embedding. Word embedding methods such as word2vec [23] and Glove [28] have been proposed and successfully applied to mining relations of medical phrases [26, 37]. More recently, there has been a surge of graph embedding methods that seek to encode structural graph information into low-dimensional dense vectors, such as Deepwalk [29], LINE [34]. Most of the embedding methods can only learn embedding vectors for

words in the corpus or nodes in the graph, and thus fail to address the OOV issue. On the other hand, some more recent inductive graph embedding works, such as Planetoid [41], GraphSAGE [12], and SEANO [19], could generate embeddings for nodes that are unobserved in the training phase by utilizing their node features (e.g., text attributes). *However, most of them assume the neighborhood of those unseen nodes is known, which is not the case for our OOV issue as the real contexts of an OOV term are unknown.* Since Planetoid [41] can generate node embeddings based on node features such as character sequence encoding vectors, it can handle the OOV issue and is chosen as a baseline model.

Synonym Discovery. A variety of methods have been proposed to detect synonyms of medical terms, ranging from utilizing lexical patterns [39] and clustering [21] to the distributional semantics models [11]. There are some more recent works on automatic synonym discovery [30, 31, 37, 42]. For example, Wang et al. [37] try to learn better embeddings for terms in medical corpora by incorporating their semantic types and then build a linear classifier to decide whether a pair of medical terms is synonyms or not. Qu et al. [30] combine distributional and pattern based methods for automatic synonym discovery. However, many aforementioned models focus on finding synonyms based on raw texts information, which is not suitable for our privacy-aware clinical data. In addition, nearly all methods could only find synonyms for terms that appear in the training corpus and, thus cannot address the OOV query terms.

6 CONCLUSION

In this paper, we study synonym discovery on privacy-aware clinical data, which is a new yet practical setting and consumes less sensitive information to discover synonyms. We propose a novel and effective framework named SurfCon that considers both the surface form information and the global context information, can handle both InV and OOV query terms, and substantially outperforms various baselines on real-world datasets. As future work, we will extend SurfCon to infer more semantic relationships (besides synonymity) between terms and test it on more real-life datasets.

ACKNOWLEDGMENTS

This research was sponsored in part by the Patient-Centered Outcomes Research Institute Funding ME-2017C1-6413, the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, and Ohio Supercomputer Center [6]. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S.Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

REFERENCES

[1] M. Ballesteros, C. Dyer, and N. A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *EMNLP*.
[2] A. L. Beam, B. Kompa, I. Fried, N. P. Palmer, X. Shi, T. Cai, and I. S. Kohane. 2018. Clinical Concept Embeddings Learned from Massive Sources of Medical Data. *arXiv preprint arXiv:1804.01486* (2018).
[3] O. Bodenreider. 2004. The unified medical language system (UMLS): integrating biomedical terminology. *Nucleic acids research* 32, suppl_1 (2004), D267–D270.
[4] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. 2016. Enriching word vectors with subword information. *TACL* (2016).
[5] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *ICML*.
[6] Ohio Supercomputer Center. 1987. Ohio Supercomputer Center. http://osc.edu/ark:/19495/f5s1ph73
[7] D. A. Dorr, W.F. Phillips, S. Phansalkar, S. A. Sims, and J. F. Hurdle. 2006. Assessing the difficulty and time cost of de-identification in clinical narratives. *Methods of information in medicine* (2006).
[8] S. G. Finlayson, P. LePendu, and N. H. Shah. 2014. Building the graph of medicine from millions of clinical narratives. *Scientific data* 1 (2014), 140032.
[9] S. L Garfinkel. 2015. De-identification of personal information. *NISTIR* (2015).
[10] W. H. Gomaa and A. A. Fahmy. 2013. A survey of text similarity approaches. In *IJCA*.
[11] M. Hagiwara, Y. Ogawa, and K. Toyama. 2009. Supervised synonym acquisition using distributional features and syntactic patterns. *IMT* (2009).
[12] W. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.
[13] K. Hashimoto, Y. Tsuruoka, R. Socher, and o. 2017. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. In *ACL*.
[14] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. 2016. Character-Aware Neural Language Models.. In *AAAI*.
[15] D. P. Kingma and J. Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.
[16] P. LePendu, S. V. Iyer, C. Fairon, and N. H. Shah. 2012. Annotation analysis for testing drug safety signals using unstructured clinical notes. In *Journal of biomedical semantics*, Vol. 3. BioMed Central, S5.
[17] O. Levy and Y. Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *ACL*.
[18] O. Levy and Y. Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NeurIPS*.
[19] J. Liang, P. Jacobs, J. Sun, and S. Parthasarathy. 2018. Semi-supervised embedding in attributed networks with outliers. In *SDM*.
[20] H. J. Lowe, T. A. Ferris, P. M. Hernandez, and S. C. Weber. 2009. STRIDE–An integrated standards-based translational research informatics platform. In *AMIA*.
[21] Y. Matsuo, T. Sakaki, and K. Uchiyama. 2006. Graph-based word clustering using a web search engine. In *EMNLP*.
[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean. 2013. Efficient estimation of word representations in vector space. *arXiv:1301.3781* (2013).
[23] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*.
[24] J. Mueller and A. Thyagarajan. 2016. Siamese Recurrent Architectures for Learning Sentence Similarity.. In *AAAI*.
[25] P. Neculoiu, M. Versteegh, and M. Rotaru. 2016. Learning text similarity with siamese recurrent networks. In *Workshop on Representation Learning for NLP*.
[26] S. V. Pakhomov, G. Finley, R. McEwan, Y. Wang, and G. B. Melton. 2016. Corpus domain effects on distributional semantic modeling of medical terms. *Bioinformatics* 32, 23 (2016), 3635–3644.
[27] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, et al. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.
[28] J. Pennington, R. Socher, and C. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.
[29] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*.
[30] M. Qu, X. Ren, and J. Han. 2017. Automatic synonym discovery with knowledge bases. In *KDD*.
[31] J. Shen, R. Lv, X. Ren, M. Vanni, B. Sadler, and J. Han. 2019. Mining Entity Synonyms with Efficient Neural Set Generation. In *AAAI*.
[32] A. Stubbs and Ö. Uzuner. 2015. Annotating longitudinal clinical narratives for de-identification: The 2014 i2b2/UTHealth corpus. *Journal of biomedical informatics* 58 (2015), S20–S29.
[33] C. N. Ta, M. Dumontier, G. Hripcsak, N. P. Tatonetti, and C. Weng. 2018. Columbia Open Health Data, clinical concept prevalence and co-occurrence from electronic health records. *Scientific data* 5 (2018), 180273.
[34] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. 2015. Line: Large-scale information network embedding. In *WWW*.
[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. Attention is all you need. In *NeurIPS*.
[36] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. 2018. Graph attention networks. In *ICLR*.
[37] C. Wang, L. Cao, and B. Zhou. 2015. Medical synonym extraction with concept space models. In *IJCAI*.
[38] Q. Wang, B. Wang, and L. Guo. 2015. Knowledge Base Completion Using Embeddings and Rules.. In *IJCAI*.
[39] J. Weeds, D. Weir, and D. McCarthy. 2004. Characterising measures of lexical distributional similarity. In *COLING*.
[40] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. 2016. Charagram: Embedding words and sentences via character n-grams. In *EMNLP*.
[41] Z. Yang, W. W. Cohen, and R. Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*.
[42] C. Zhang, Y. Li, N. Du, W. Fan, and P. S. Yu. 2018. SynonymNet: Multi-context Bilateral Matching for Entity Synonyms. *arXiv preprint arXiv:1901.00056* (2018).

Reinforced Dynamic Reasoning for Conversational Question Generation

Boyuan Pan[1]*, **Hao Li**[1], **Ziyu Yao**[2], **Deng Cai**[1,3], **Huan Sun**[2]
[1]State Key Lab of CAD&CG, Zhejiang University
[2]The Ohio State University
[3]Alibaba-Zhejiang University Joint Institute of Frontier Technologies
{panby, haolics, dcai}@zju.edu.cn
{yao.470, sun.397}@osu.edu

Abstract

This paper investigates a new task named *Conversational Question Generation* (CQG) which is to generate a question based on a passage and a conversation history (i.e., previous turns of question-answer pairs). CQG is a crucial task for developing intelligent agents that can drive question-answering style conversations or test user understanding of a given passage. Towards that end, we propose a new approach named Reinforced Dynamic Reasoning (ReDR) network, which is based on the general encoder-decoder framework but incorporates a reasoning procedure in a *dynamic* manner to better understand what has been asked and what to ask next about the passage. To encourage producing meaningful questions, we leverage a popular question answering (QA) model to provide feedback and fine-tune the question generator using a reinforcement learning mechanism. Empirical results on the recently released CoQA dataset demonstrate the effectiveness of our method in comparison with various baselines and model variants. Moreover, to show the applicability of our method, we also apply it to create multi-turn question-answering conversations for passages in SQuAD.

1 Introduction

In this work, we study a novel task of *conversational question generation* (CQG) which is given a passage and a conversation history (i.e., previous turns of question-answer pairs), to generate the next question.

CQG is an important task in its own right for measuring the ability of machines to lead a question-answering style conversation. It can serve as an essential component of intelligent social bots or tutoring systems, asking meaningful and coherent questions to engage users or test student understanding about a certain topic. On the other hand, as shown in Figure 1, large-scale high-quality conversational question answering (CQA) datasets such as CoQA (Reddy et al., 2018) and QuAC (Choi et al., 2018) can help train models to answer sequential questions. However, manually creating such datasets is quite costly, e.g., CoQA spent 3.6 USD per passage on crowdsourcing for conversation collection, and automatic CQG can potentially help reduce the cost, especially when there are a large set of passages available.

In recent years, automatic question generation (QG), which aims to generate natural questions based on a certain type of data sources including structured knowledge bases (Serban et al., 2016b; Guo et al., 2018) and unstructured texts (Rus et al.,

Shelly is in second grade. She is a new student at her school. Shelly's family has lived in many different places. Shelly was born in Florida. Her family moved to Tennessee when she was two years old. When she was four years old, they moved to Texas. They moved from there to Arizona, where they now live.

Q1: What grade is Shelly in ?
A1: second
R1: Shelly is in second grade.

Q2: Was she a new student ?
A2: Yes
R2: She is a new student at her school.

Q3: Where did she move at 2 years old ?
A2: Tennessee
R3: Her family moved to Tennessee when she was two years old.

Figure 1: An example from the CoQA dataset. Each turn contains a question (Q) and an answer (A). The dataset also provides a rationale (R) (i.e., a text span from the passage) to support each answer.

*Work done while visiting the Ohio State University.

2010; Heilman and Smith, 2010; Du et al., 2017; Du and Cardie, 2018), has been widely studied. However, previous works mainly focus on generating standalone and independent questions based on a given passage. To the best of our knowledge, we are the first to explore CQG, i.e., generating the next question based on a passage and *a conversation history*.

Comparing with previous QG tasks, CQG needs to take into account not only the given passage, but also the conversation history, and is potentially more challenging as it requires a deep understanding of what has been asked so far and what information should be asked for the next round, in order to make a coherent conversation.

In this paper, we present a novel framework named *Reinforced Dynamic Reasoning* (ReDR) network. Inspired by the recent success of reading comprehension models (Xiong et al., 2017; Seo et al., 2017), ReDR adapts their reasoning procedure (which encodes the knowledge of the passage and the conversation history based on a coattention mechanism) and moreover *dynamically* updates the encoding representation based on a soft decision maker to generate a coherent question. In addition, to encourage ReDR to generate meaningful and interesting questions, ideally, one may employ humans to provide feedback, but as widely acknowledged, involving humans in the loop for training models can be very costly. Therefore, in this paper, we leverage a popular and effective reading comprehension (or QA) model (Chen et al., 2017) to predict the answer to a generated question and use its answer quality (which can be seen as a proxy for real human feedback) as rewards to fine-tune our model based on a reinforcement learning mechanism (Williams, 1992).

Our contributions are summarized as follows:

- We introduce a new task of *Conversational Question Generation* (CQG), which is crucial for developing intelligent agents to drive question-answering style conversations and can potentially provide valuable datasets for future relevant research.
- We propose a new and effective framework for CQG, which is equipped with a dynamic reasoning component to generate a conversational question and is further fine-tuned via a reinforcement learning mechanism.
- We show the effectiveness of our method using the recent CoQA dataset. Moreover, we show its wide applicability by using it to create multi-turn QA conversations for passages in SQuAD (Rajpurkar et al., 2016).

2 Task Definition

Formally, we define the task of *Conversational Question Generation* (CQG) as: Given a passage X and the previous turns of question-answer pairs $\{(q_1, a_1), (q_2, a_2), ..., (q_{k-1}, a_{k-1})\}$ about X, CQG aims to generate the next question q_k that is related to the given passage and coherent with the previous questions and answers, i.e.,

$$q_k = \arg\max_{q_k} P(q_k|X, q_{<k}, a_{<k}) \quad (1)$$

where $P(q_k|X, q_{<k}, a_{<k})$ is a conditional probability of generating the question q_k.

3 Methodology

We show our proposed framework named *Reinforced Dynamic Reasoning (ReDR)* network in Figure 2. Since a full passage is usually too long and makes it hard to focus on the most relevant information for generating the next question, our method first selects a text span from the passage as the rationale at each conversation turn, and then dynamically models the reasoning procedure for encoding the conversation history and the selected rationale, before finally decoding the next question.

3.1 Rationale Selection

We simply set each sentence in the passage as the corresponding rationale for each turn of the conversation. When experimenting with CoQA, we use the rationale span provided in the dataset. Besides for simplicity and efficiency, another reason that we adopt this rule-based method is that previous research demonstrated that the transition of the dialog attention is smooth (Reddy et al., 2018; Choi et al., 2018), meaning that earlier questions in a conversation are usually answerable by the preceding part of the passage while later questions tend to focus on the ending part of the passage. The selected rationale is then leveraged by subsequent modules for question generation.

3.2 Encoding & Reasoning

At each turn k, we denote the conversation history as a sequence of m tokens, i.e., $c =$

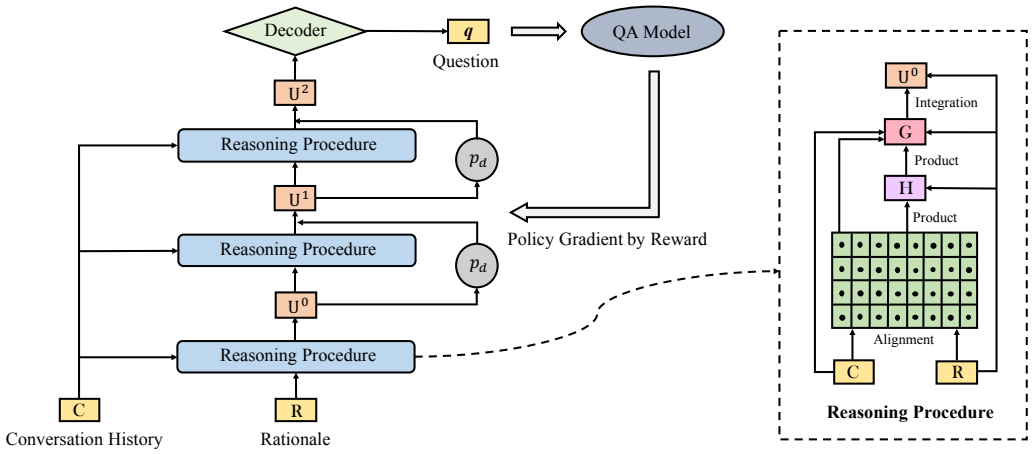


Figure 2: Overview of our Reinforced Dynamic Reasoning (ReDR) network. The reasoning mechanism iteratively reads the conversation history and at each iteration, its output is dynamically combined with the previous encoding representation through a soft decision maker (p_d) as the new encoding representation, which is fed into the next iteration. The model is finally fine-tuned by the reward defined by the quality of the answer predicted from a QA model.

$\{c_1, c_2, ..., c_m\}$, which concatenates the previous questions and answers $<q_1, a_1, ..., q_{k-1}, a_{k-1}>$, and represent the rationale as a sequence of n tokens, i.e., $r = \{r_1, r_2, ..., r_n\}$. As mentioned earlier, different from previous question generation tasks, we have two knowledge sources (i.e., the conversation history and the rationale) as the inputs. A good encoding of them is crucial for task performance and might involve a reasoning procedure across previous question-answer pairs and the selected rationale for determining the next question. We feed them respectively into a bi-directional LSTM and obtain their contextual representations $\mathbf{C} \in R^{d\times m}$ and $\mathbf{R} \in R^{d\times n}$. Inspired by the coattention reasoning mechanism in previous reading comprehension works (Xiong et al., 2017; Seo et al., 2017; Pan et al., 2017), we compute an alignment matrix of $\mathbf{C}$ and $\mathbf{R}$ to link and fuse the information flow: $\mathbf{S} = \mathbf{R}^\top \mathbf{C} \in R^{n\times m}$. We normalize this alignment matrix column-wise (i.e., softmax($\mathbf{S}$)) to obtain the relevance degree of each token in the conversation history to the whole rationale. The new representation of the conversation history w.r.t. the rationale is obtained via:

$$\mathbf{H} = \mathbf{R} \cdot \text{softmax}(\mathbf{S}) \in R^{d\times m} \quad (2)$$

Similarly, we compute the attention over the conversation history for each word in the rationale via $\text{softmax}(\mathbf{S}^\top)$ and obtain the context-dependent representation of the rationale by $\mathbf{C} \cdot \text{softmax}(\mathbf{S}^\top)$. In addition, as in (Xiong et al., 2017), we also consider the above new representation of the conversation history and map it to the space of rationale encodings via $\mathbf{H} \cdot \text{softmax}(\mathbf{S}^\top)$, and finally obtain the co-dependent representation of the rationale and the conversation history:

$$\mathbf{G} = [\mathbf{C}; \mathbf{H}] \cdot \text{softmax}(\mathbf{S}^\top) \in R^{2d\times n} \quad (3)$$

where $[;]$ means concatenation across row dimension. To deeply capture the interaction between the rationale and the conversation history, we feed the co-dependent representation $\mathbf{G}$ combined with the rationale $\mathbf{R}$ into an *integration model* instantiated by a bi-directional LSTM:

$$\mathbf{u}_i^0 = \text{BiLSTM}(\mathbf{u}_{i-1}^0, \mathbf{u}_{i+1}^0, [\mathbf{G}_i; \mathbf{R}_i]) \in R^d \quad (4)$$

We define the reasoning process in our paper as Eqn. (2-4), and now obtain a matrix $\mathbf{U}^0 = [\mathbf{u}_1^0, \mathbf{u}_2^0, ..., \mathbf{u}_n^0]$ as the encoding representation after one-layer reasoning procedure, which can be fed into the decoder subsequently.

3.3 Dynamic Reasoning

Oftentimes the conversation history is very informative and complicated, and one single layer of reasoning may be insufficient to comprehend the subtle relationship among the rationale, the conversation history, and the to-be-generated question. Therefore, we propose a dynamic reasoning procedure to iteratively update the encoding representation. We regard $\mathbf{U}^0$ as a new representation

of the rationale and input it to the next layer of reasoning together with $\mathbf{C}$:

$$\widetilde{\mathbf{U}}^1 = F_{reason}(\mathbf{U}^0, \mathbf{C}) \tag{5}$$

where F_{reason} is the reasoning procedure (Eqn. 2-4), and $\widetilde{\mathbf{U}}^1$ is the hidden states of the BiLSTM integration model at the next reasoning layer. To effectively learn what information in $\widetilde{\mathbf{U}}^1$ and $\mathbf{U}^0$ is relevant to keep, we use a soft *decision maker* to determine their weights:

$$\begin{aligned} \mathbf{U}^1 &= \mathbf{p}_d \odot \mathbf{U}^0 + (\mathbf{e}_1 - \mathbf{p}_d) \odot \widetilde{\mathbf{U}}^1 \\ \mathbf{p}_d &= \sigma(\mathbf{w}_{\mathrm{u}}^\top \mathbf{U}^0 + \mathbf{w}_{\mathrm{g}}^\top \mathbf{G} + \mathbf{w}_{\mathrm{r}}^\top \mathbf{R} + \mathbf{b}) \end{aligned} \tag{6}$$

where $\mathbf{e}_1$ is an all-ones vector, and $\mathbf{w}_u, \mathbf{w}_g, \mathbf{w}_r, \mathbf{b}$ are trainable parameters. $\mathbf{p}_d \in R^n$ is the decision maker, used as a soft switch to choose between different levels of reasoning. $\mathbf{U}^1$ is the representation to be used for the next layer of reasoning. This iterative procedure halts when a maximum number of reasoning layers N is reached ($N \geq 1$). The final representation $\mathbf{U}^N$ is fed into the decoder.

3.4 Decoding

The decoder generates a word by sampling from the probability $P_{gen}(y_t|y_{<t}, c, r)$ which can be computed via:

$$\begin{aligned} P_{gen}(y_t|y_{<t}, c, r) &= \mathrm{MLP}(\mathbf{o}_t, \mathbf{v}_t) \\ \mathbf{o}_t &= \mathrm{LSTM}(\mathbf{o}_{t-1}, \mathrm{Emb}(y_{t-1}), \mathbf{v}_{t-1}) \end{aligned} \tag{7}$$

where MLP stands for a standard multilayer perceptron network, y_t is the t-th word in the generated question, $\mathbf{o}_t$ is the hidden state of the decoder at time step t, and $\mathrm{Emb}(\cdot)$ indicates the word embedding. $\mathbf{v}_t$ is an attentive read of the encoding representation: $\mathbf{v}_t = \sum_{i=1}^n \alpha_{t,i} \mathbf{u}_i^N$, where the weight $\alpha_{t,i} \in (0, 1)$ is scored by another $\mathrm{MLP}(\mathbf{o}_t, \mathbf{u}_i^N)$ network.

Observing that a question may share common words with the rationale that it is based on and inspired by the widely adopted copy mechanism (Gu et al., 2016; See et al., 2017), we also apply a pointer network for the generator to copy words from the rationale. Now the probability of generating target word y_t becomes:

$$P(y_t|y_{<t}, c, r) = \lambda P_{gen}(y_t) + (1 - \lambda) P_{pt}(y_t) \tag{8}$$

where $P_{gen}(y_t)$=$P_{gen}(y_t|y_{<t}, c, r)$ is defined earlier, $P_{pt}(y_t) = \sum_{i:r_i=y_t} \alpha_{t,i}$ is the probability of copying word y_t from r (only if r contains y_t), and λ is the weight to balance the two:

$$\lambda = \sigma(\mathbf{w}_v^\top \mathbf{v}_t + \mathbf{w}_o^\top \mathbf{o}_t + \mathbf{w}_y^\top \mathrm{Emb}(y_{t-1}) + \mathbf{b}_{pt}) \tag{9}$$

where $\mathbf{w}_v^\top$, $\mathbf{w}_o^\top$, $\mathbf{w}_y^\top$ and $\mathbf{b}_{pt}$ are to be learnt. To optimize all parameters in ReDR, we adopt the maximum likelihood estimation (MLE) approach, i.e., maximizing the summed log likelihood of words in a target question.

3.5 Reinforcement Learning for Fine-tuning

As shown by recent datasets like CoQA and QuAC, human-created questions tend to be meaningful and interesting. For example, in Figure 1, given the second rationale R2 "She is a new student at her school", humans tend not to ask "Where is she?", and similarly given R3, they usually do not create the question "What happened?". Although both are legitimate questions, they tend to be less interesting and meaningful compared with the human-created ones shown in Figure 1. The interestingness or meaningfulness of a question is subjective and hard to define, automatically measuring which is a difficult problem itself. Ideally, one can involve humans in the loop to judge the generated question and provide feedback, but it can be very costly, if not impossible.

Driven by such observations, we use the REINFORCE (Williams, 1992) algorithm and adopt one of the state-of-the-art reading comprehension models DrQA (Chen et al., 2017) as a substitute for humans to provide feedback to the question generator. DrQA answers a question based on the given passage and has achieved a competitive performance on CoQA (Reddy et al., 2018). During training, we apply DrQA to answer a generated question, and compare its answer with the human-provided answer (which is associated with the same rationale for generating the question)[1]. If the answers match well with each other, we regard our generator produces a meaningful question since it asks about the same thing as humans do, and will assign high rewards to such questions.

Formally, we minimize the negative expected reward for a generated question:

$$J_{RL} = -\mathbb{E}_{q\sim\pi(q|r,c)}[R(a, a^*)] \tag{10}$$

where $\pi(q|r, c) = \prod_t P(y_t|y_{<t}, c, r)$ is the action policy defined in Eqn. (8) for producing question

[1]We use the CoQA dataset for training and such information is available as shown in Figure 1.

Dataset	Passages	QA Pairs	Turns per Passage
Training	7199	10.8k	15.0
Dev	500	8.0k	15.9

Table 1: Statistics of the CoQA dataset.

q given rationale r and conversation history c, and $R(a, a^*)$ is the reward function defined by the F1 score[2] between the DrQA predicted answer a and the human-provided answer a^*. For computational efficiency concerns, during training, we make sure that the ground-truth question is in the sampling pool and use beam search to generate 5 more questions.

Note that besides providing rewards for fine-tuning our generator, DrQA model also serves another purpose: When applying our framework to any passage, we can use DrQA to produce an answer to the currently generated question so that the conversation history can be updated for the next-turn of question generation. In addition, our framework is not limited to DrQA and other more advanced QA models can apply as well.

4 Experiments

4.1 Dataset

We use the CoQA dataset[3] (Reddy et al., 2018) to experiment with our ReDR and baseline methods. CoQA contains text passages from diverse domains, conversational questions and answers developed for each passage, as well as rationales (i.e., text spans extracted from given passages) to support answers. The dataset consists of 108k questions in the training set and 8k questions in the development (dev) set with a large hidden test set for competition purpose, and our results are shown on the dev set.

4.2 Baselines

As discussed earlier, CQG has been under-investigated so far, and there are few existing baselines for our comparison. Because of their high relevance with our task as well as their superior performance demonstrated by previous works, we choose to compare with the following models:

Seq2Seq (Sutskever et al., 2014) is a basic encoder-decoder sequence learning system, which has been widely used for machine translation (Luong et al., 2015) and dialogue generation (Wen et al., 2017). We concatenate the rationale and the conversation history as the input sequence in our setting.

NQG (Du et al., 2017) is a strong attention-based neural network approach for question generation task. The input is the same as the above Seq2Seq model.

4.3 Implementation Details

Our word embeddings are initialized by `glove.840B.300d` (Pennington et al., 2014). We set the LSTM hidden unit size to 500 and set the number of layers of LSTMs to 2 in both the encoder and the decoder. Optimization is performed using stochastic gradient descent (SGD), with an initial learning rate of 1.0. The learning rate starts decaying at the step 15000 with a decay rate of 0.95 for every 5000 steps. The mini-batch size for the update is set at 64. We set the dropout (Srivastava et al., 2014) ratio as 0.3 and the beam size as 5. The maximum number of iterations for the dynamic reasoning is set to be 3. Since the CoQA contains abstractive answers, we apply DrQA as our question answering model and follow Yatskar (2018) to separately train a binary classifier to produce "yes" or "no" for yes/no questions[4]. Code is available at `https://github.com/ZJULearning/ReDR`.

4.4 Automatic Evaluation

Metrics We follow previous question generation work (Xu et al., 2017; Du et al., 2017) to use BLEU[5] (Papineni et al., 2002) and ROUGE-L (Lin, 2004) to measure the *relevance* between the generated question and the ground-truth one. To evaluate the *diversity* of the generated questions, we follow (Li et al., 2016a) to calculate Dist-n (n=1,2), which is the proportion of unique n-grams over the total number of n-grams in the generated questions for all passages, and (Zhang et al., 2018) to use the Ent-n (n=4) metric, which reflects how evenly the n-gram distribution is over all generated questions. For all the metrics, the larger they are,

[2] F1 score is the common evaluation metric for QA and is defined as the harmonic mean of precision and recall.

[3] https://stanfordnlp.github.io/coqa/

[4] Our modified DrQA model achieves 68.8 F1 scores on the CoQA dev set.

[5] We adopt the 4th smoothing technique as proposed in (Chen and Cherry, 2014) for short text generation.

Models	Relevance		Diversity		
	BLEU	RG-L	Dist-1	Dist-2	Ent-4
Vanilla Seq2Seq Model	7.64	26.68	0.010	0.034	3.370
NQG (Du et al., 2017)	13.97	31.75	0.017	0.068	6.518
With 1 Layer Reasoning, no RL	16.13	32.24	0.053	0.171	7.862
With 2 Layer Reasoning, no RL	17.85	33.06	0.062	0.216	8.285
With 3 Layer Reasoning, no RL	17.42	32.88	0.061	0.205	8.247
With Dynamic Reasoning, no RL	19.10	33.57	0.064	0.220	8.304
Reinforced Dynamic Reasoning (ReDR)	**19.69**	**34.05**	**0.069**	**0.225**	**8.367**

Table 2: Quantitative evaluation for conversational question generation using CoQA dataset.

the more relevant or diverse the generated questions are.

Results and Analysis Table 2 shows the performance of various models on the CoQA dataset. As we can see, our model ReDR and its variants perform much better than the baselines, which indicates that the reasoning procedure can significantly boost the quality of the encoding representations and thus improve the question generation performance.

To investigate the effect of the reasoning procedure and fine-tuning in our model design, we also conduct an ablation study: (1) We first test our model with only one layer of reasoning, i.e., directly feeding the encoding representation $\mathbf{U}^0$ into the decoder. The results drop a lot on all the metrics, which indicates that there is abundant semantic information in the input text so the multi-layer reasoning is necessary. (2) We then augment our model with two or three layers of reasoning but without the decision maker $\mathbf{p}_d$. In other words, we directly use the hidden states of the integration LSTM as the input to the next reasoning layer (formally, $U^j = \tilde{U}^j$). We can see that the performance of our model increases with a two-layer reasoning while decreases with a three-layer reasoning. We conjecture that the two-layer reasoning network is saturated for most of the input text sequences, thus directly adding a layer of network for all the input text seems not optimal. (3) When we add the decision maker to dynamically compute the encoding representations, the results are greatly improved, which demonstrates that using a dynamic procedure can distribute proper weight of each layer to the input sequences in different lengths and amount of information. (4) Finally, we fine-tune the model with the reinforcement learning framework, and the results show that using the answer quality as the reward is helpful for generating better questions.

	NQG	ReDR	Human
Naturalness	1.94	1.92	2.14
Relevance	1.16	2.02	2.82
Coherence	1.12	1.94	2.94
Richness	1.16	2.30	2.54
Answerability	1.18	1.86	2.96

Table 3: Human evaluation results on CoQA. "Human" in the table means the original human-created questions in CoQA.

4.5 Human Evaluation

We conduct human evaluation to measure the quality of generated questions. We randomly sampled 50 questions along with their conversation history and the passage, and consider 5 aspects: *Naturalness*, which indicates the grammaticality and fluency; *Relevance*, which indicates the connection with the topic of the passage; *Coherence*, which measures whether the generated question is coherent with the conversation history; *Richness*, which measures the amount of information contained in the question. *Answerability*, which indicates whether the question is answerable based on the passage. For each sample, 5 people [6] are asked to rank three questions (the ReDR question, the NQG question and the human-created question) by assigning each a score from {1,2,3} (the higher, the better). For each aspect, we show the average score across the five annotators on all samples.

Table 3 shows the results of human evaluation. We can see that our method almost outperforms NQG in all aspects. For *Naturalness*, the three

[6] All annotators are native English speakers.

Category	NQG	ReDR	Human
Question Type			
"what" Question	0.45	0.42	0.35
"which" Question	0.01	0.01	0.02
"when" Question	0.07	0.05	0.04
"where" Question	0.08	0.06	0.07
"who" Question	0.06	0.22	0.15
"why" Question	0.15	0.03	0.03
yes/no Question	0.08	0.07	0.21
Linguistic Feature			
Question Length	4.05	5.34	6.48
Explicit Coref.	0.51	0.53	0.47
Implicit Coref.	0.32	0.19	0.19

Table 4: Linguistic statistics for the generated questions and the human annotated questions in CoQA.

methods obtain the similar scores, which is probably because that the most generated questions are short and fluent, makes them have no significant difference on this aspect. We also observe that on the *Relevance*, *Coherence* and *Answerability* aspects, there is an obvious gap between the generative models and human annotation. This indicates that the contextual understanding is still a challenging problem for the task of the conversational question generation.

4.6 Linguistic Analysis

We further analyze the generated questions in terms of their linguistic features and constitutions in Table 4, from which we draw three observations: (1) Overall, the distribution of the major types of questions generated by ReDR is closer to human-created questions, in comparison with NQG. For example, ReDR generates a large portion of "what" and "who" questions, similarly as humans. (2) We observe that NQG tends to generate many single-word questions such as "Why?" while our method successfully alleviates this problem. (3) Both ReDR and NQG generate fewer yes/no questions than humans, as a result of generating more "wh"-type of questions.

For the relationship between a question and its conversation history, following the analysis in CoQA, we randomly sample 150 questions respectively from each method and observe that about 50% questions generated by ReDR contain explicit coreference markers such as "he", "she" or "it", which is similar to the other two methods.

Once upon a time, in a barn near a farm house, there lived a little white kitten named Cotton. Cotton lived high up in a nice warm place above the barn where all of the farmer's horses slept. But Cotton wasn't alone in her little home above the barn, oh no. She shared her hay bed with her mommy and 5 other sisters...

OQ1: What color was cotton ?
A1: white
NQG: What type of animal was it ?
ReDR: **What was the animal 's name ?**

OQ2: Where did she live ?
A2: in a barn
NQG: What was it ?
ReDR: **What kind of house did she live ?**

OQ3: Did she live alone ?
A3: no
NQG: Why ?
ReDR: **Was she alone ?**

OQ4: Who did she live with?
A4: with her mommy and 5 sisters
NQG: What does she do ?
ReDR: **Who else ?**

Figure 3: Example questions generated by human (i.e., original questions denoted as OQ), NQG and our ReDR on CoQA.

However, NQG generates much more questions consisting of implicit coreference markers like "Where?" or "Who?", which can be less meaningful or not answerable as also verified in Table 3.

4.7 Case Study

In Figure 3, we show the output questions of our ReDR and NQG on an example from CoQA dataset. For the first turn, both ReDR and NQG generate a meaningful and answerable question. For the second turn, NQG generates "What was it?", which is answerable and related to the conversation history but simpler than our question "What kind of house did she live?". For the third turn, NQG generates a coherent but less meaningful question "Why?", while our method generates "Was she alone?", which is very similar to the human-created question. For the last turn, NQG produces a question that is neither coherent nor answerable, while ReDR asks a much better question "Who else?".

To show the applicability of ReDR to generate QA style conversations on any passages, we apply it to passages in the SQuAD reading comprehension dataset (Rajpurkar et al., 2016) and show an example in Figure 4. Since there are no rationales

The game's Media Day, which was typically held on the Tuesday afternoon prior to the game, was moved to the Monday evening and rebranded as super bowl opening night. The event was held on February 1, 2016 at Sap Center in San Jose. Alongside the traditional media availabilities, the event featured an opening ceremony with player introductions on a replica of the golden gate bridge …

Q1: What was held on Monday ?
A1: game's Media Day

Q2: Where ?
A2: Sap Center

Q3: What was the opening ceremony for ?
A3: player introductions

Figure 4: Our generated conversation on a SQuAD passage. The questions are generated by our ReDR and the answers are predicted by DrQA.

provided in the dataset for generating consecutive questions, we first apply our rule-based rationale selection as introduced in Section 3.1 and then generate a question based on the selected rationale and the conversation history. The answers are predicted by our modified DrQA. Figure 4 shows that our generated questions are closely related to the passage, e.g., the first question contains "Monday" and the third one mentions "opening ceremony". Moreover, we can also generate interesting questions such as "Where?" which connects to previous questions and makes a coherent conversation.

5 Related Work

Question Generation. Generating questions from various kinds of sources, such as texts (Rus et al., 2010; Heilman and Smith, 2010; Mitkov and Ha, 2003; Du et al., 2017), search queries (Zhao et al., 2011), knowledge bases (Serban et al., 2016b) and images (Mostafazadeh et al., 2016), has attracted much attention recently. Our work is most related to previous work on generating questions from sentences or paragraphs. Most early approaches are based on rules and templates (Heilman and Smith, 2010; Mitkov and Ha, 2003), while Du et al. (2017) recently proposed to generate a question by a Sequence-to-Sequence neural network model (Sutskever et al., 2014) with attention (Luong et al., 2015). Other approaches such as (Zhou et al., 2017; Subramanian et al., 2017) take into account the answer information in addition to the given sentence or paragraph. (Du and Cardie, 2018; Song et al., 2018) further modeled the surrounding paragraph-level information of the given sentence. However, most of the work focused on generating standalone questions solely based on a sentence or a paragraph. In contrast, this work explores *conversational* question generation and has to additionally consider the conversation history in order to generate a coherent question, making the task much more challenging.

Conversation Generation. Building chatbots and conversational agents has been pursued by many previous work (Ritter et al., 2011; Vinyals and Le, 2015; Sordoni et al., 2015; Serban et al., 2016a; Li et al., 2016a,b). Vinyals and Le (2015) used a Sequence-to-Sequence neural network (Sutskever et al., 2014) for generating a response given the dialog history. Li et al. (2016a) further optimized the response diversity by maximizing the mutual information between inputs and output responses. Different from these work where the response can be in any form (usually a declarative statement) and is generated solely based on the dialog history, our task is potentially more challenging as it additionally restricts the generated response to be a *follow-up question* about a given *passage*.

Conversational Question Answering (CQA). CQA aims to automatically answer a sequence of questions. It has been studied in the knowledge base setting (Saha et al., 2018; Iyyer et al., 2017) and is often framed as a semantic parsing problem. Recently released large-scale datasets (Reddy et al., 2018; Choi et al., 2018) enabled studying it in the textual setting where the information source used to answer questions is a given passage, and they inspired many significant work (Zhu et al., 2018; Huang et al., 2018; Yatskar, 2018). However, collecting such datasets has heavily relied on human efforts and can be very costly. Based on one of the most popular datasets CoQA (Reddy et al., 2018), we examine the possibility of automatically *generating* conversational questions, which can potentially reduce the data collection cost for CQA.

6 Conclusion

In this paper, we introduce the task of *Conversational Question Generation* (CQG), and propose a novel framework which achieves promising performance on the popular dataset CoQA. We in-

corporate a dynamic reasoning procedure to the general encoder-decoder model and dynamically update the encoding representations of the inputs. Moreover, we use the quality of the answers predicted by a QA model as rewards and fine-tune our model via reinforcement learning. In the future, we would like to explore how to better select the rationale for each question. Besides, it would also be interesting to consider using linguistic knowledge such as named entities or part-of-speech tags to improve the coherence of the conversation.

7 Acknowledgments

This research was sponsored in part by the Army Research Office under grant W911NF-17-1-0412, NSF Grant IIS-1815674, the National Nature Science Foundation of China (grant No. 61751307), and Ohio Supercomputer Center (Center, 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Ohio Supercomputer Center. 1987. Ohio supercomputer center.

Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 362–367.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1870–1879.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. 2018. Quac: Question answering in context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2174–2184.

Xinya Du and Claire Cardie. 2018. Harvesting paragraph-level question-answer pairs from wikipedia. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1907–1917.

Xinya Du, Junru Shao, and Claire Cardie. 2017. Learning to ask: Neural question generation for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1342–1352.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 1, pages 1631–1640.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question generation from sql queries improves neural semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1597–1607.

Michael Heilman and Noah A Smith. 2010. Good question! statistical ranking for question generation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 609–617. Association for Computational Linguistics.

Hsin-Yuan Huang, Eunsol Choi, and Wen-tau Yih. 2018. Flowqa: Grasping flow in history for conversational machine comprehension. *arXiv preprint arXiv:1810.06683*.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1821–1831.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016a. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. 2016b. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Ruslan Mitkov and Le An Ha. 2003. Computer-aided generation of multiple-choice tests. In *Proceedings of the HLT-NAACL 03 workshop on Building educational applications using natural language processing-Volume 2*, pages 17–22. Association for Computational Linguistics.

Nasrin Mostafazadeh, Ishan Misra, Jacob Devlin, Margaret Mitchell, Xiaodong He, and Lucy Vanderwende. 2016. Generating natural questions about an image. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1802–1813.

Boyuan Pan, Hao Li, Zhou Zhao, Bin Cao, Deng Cai, and Xiaofei He. 2017. Memen: Multi-layer embedding with memory networks for machine comprehension. *arXiv preprint arXiv:1707.09098*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2383–2392.

Siva Reddy, Danqi Chen, and Christopher D Manning. 2018. Coqa: A conversational question answering challenge. *arXiv preprint arXiv:1808.07042*.

Alan Ritter, Colin Cherry, and William B Dolan. 2011. Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*, pages 583–593. Association for Computational Linguistics.

Vasile Rus, Brendan Wyse, Paul Piwek, Mihai Lintean, Svetlana Stoyanchev, and Christian Moldovan. 2010. The first question generation shared task evaluation challenge. In *Proceedings of the 6th International Natural Language Generation Conference*.

Amrita Saha, Vardaan Pahuja, Mitesh M Khapra, Karthik Sankaranarayanan, and Sarath Chandar. 2018. Complex sequential question answering: Towards learning to converse over linked question answer pairs with a knowledge graph. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. *ICLR*.

Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. 2016a. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Iulian Vlad Serban, Alberto García-Durán, Caglar Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio. 2016b. Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 588–598.

Linfeng Song, Zhiguo Wang, Wael Hamza, Yue Zhang, and Daniel Gildea. 2018. Leveraging context information for natural question generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, volume 2, pages 569–574.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015. A neural network approach to context-sensitive generation of conversational responses. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 196–205.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Sandeep Subramanian, Tong Wang, Xingdi Yuan, Saizheng Zhang, Yoshua Bengio, and Adam Trischler. 2017. Neural models for key phrase detection and question generation. *arXiv preprint arXiv:1706.04560*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Oriol Vinyals and Quoc Le. 2015. A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gasic, Lina M Rojas Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. 2017. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 438–449.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Caiming Xiong, Victor Zhong, and Richard Socher. 2017. Dynamic coattention networks for question answering. *ICLR*.

Zhen Xu, Bingquan Liu, Baoxun Wang, SUN Chengjie, Xiaolong Wang, Zhuoran Wang, and Chao Qi. 2017. Neural response generation via gan with an approximate embedding layer. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 617–626.

Mark Yatskar. 2018. A qualitative comparison of coqa, squad 2.0 and quac. *arXiv preprint arXiv:1809.10735*.

Yizhe Zhang, Michel Galley, Jianfeng Gao, Zhe Gan, Xiujun Li, Chris Brockett, and Bill Dolan. 2018. Generating informative and diverse conversational responses via adversarial information maximization. In *Advances in Neural Information Processing Systems*, pages 1815–1825.

Shiqi Zhao, Haifeng Wang, Chao Li, Ting Liu, and Yi Guan. 2011. Automatically generating questions from queries for community-based question answering. In *Proceedings of 5th international joint conference on natural language processing*, pages 929–937.

Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. 2017. Neural question generation from text: A preliminary study. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 662–671. Springer.

Chenguang Zhu, Michael Zeng, and Xuedong Huang. 2018. Sdnet: Contextualized attention-based deep network for conversational question answering. *arXiv preprint arXiv:1812.03593*.

CoaCor: Code Annotation for Code Retrieval with Reinforcement Learning

Ziyu Yao
The Ohio State University
yao.470@osu.edu

Jayavardhan Reddy Peddamail
The Ohio State University
peddamail.1@osu.edu

Huan Sun
The Ohio State University
sun.397@osu.edu

ABSTRACT

To accelerate software development, much research has been performed to help people understand and reuse the huge amount of available code resources. Two important tasks have been widely studied: *code retrieval*, which aims to retrieve code snippets relevant to a given natural language query from a code base, and *code annotation*, where the goal is to annotate a code snippet with a natural language description. Despite their advancement in recent years, the two tasks are mostly explored separately. In this work, we investigate a novel perspective of *Code annotation for Code retrieval* (hence called "CoaCor"), where a code annotation model is trained to generate a natural language annotation that can represent the semantic meaning of a given code snippet and can be leveraged by a code retrieval model to better distinguish relevant code snippets from others. To this end, we propose an effective framework based on reinforcement learning, which explicitly encourages the code annotation model to generate annotations that can be used for the retrieval task. Through extensive experiments, we show that code annotations generated by our framework are much more detailed and more useful for code retrieval, and they can further improve the performance of existing code retrieval models significantly.[1]

CCS CONCEPTS

• **Information systems** → **Novelty in information retrieval**; **Summarization**; • **Software and its engineering**; • **Computing methodologies** → *Reinforcement learning*; *Markov decision processes*; Neural networks;

KEYWORDS

Code Annotation; Code Retrieval; Reinforcement Learning

ACM Reference Format:
Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. 2019. CoaCor: Code Annotation for Code Retrieval with Reinforcement Learning. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3308558.3313632

1 INTRODUCTION

Software engineering plays an important role in modern society. Almost every aspect of human life, including health care, education, transportation and web security, depends on reliable software [1]. Unfortunately, developing and maintaining large code bases are very costly. Understanding and reusing billions of lines of code in online open-source repositories can significantly speed up the software development process. Towards that, *code retrieval* (CR) and *code annotation* (CA) are two important tasks that have been widely studied in the past few years [1, 13, 19, 21, 58], where the former aims to retrieve relevant code snippets based on a natural language (NL) query while the latter is to generate natural language descriptions to describe what a given code snippet does.

Most existing work [2, 13, 19, 22, 58, 60] study either code annotation or code retrieval individually. Earlier approaches for code retrieval drew inspiration from the information retrieval field [14, 17, 23, 32] and suffered from surface form mismatches between natural language queries and code snippets [6, 34]. More recently, advanced deep learning approaches have been successfully applied to both code retrieval and code annotation [1, 2, 9, 13, 19–22, 31, 58, 60]. For example, the code retrieval model proposed by Gu et al. [13] utilized two deep neural networks to learn the vector representation of a natural language query and that of a code snippet respectively, and adopted cosine similarity to measure their matching degree. For code annotation, Iyer et al. [21] and Hu et al. [19] utilized encoder-decoder models with an attention mechanism to generate an NL annotation for a code snippet. They aim to generate annotations similar to the human-provided ones, and therefore trained the models using the standard maximum likelihood estimation (MLE) objective. For the same purpose, Wan et al. [58] trained the code annotation model in a reinforcement learning (RL) framework with reward being the BLEU score [41], which measures n-gram matching precision between the currently generated annotation and the human-provided one.

In this work, we explore a novel perspective - code annotation *for* code retrieval (CoaCor), which is to generate an NL annotation for a code snippet so that the generated annotation *can be used for code retrieval* (i.e., *can represent the semantic meaning of a code snippet and distinguish it from others w.r.t. a given NL query in the code retrieval task*). As exemplified by [56], such an annotation can be taken as the representation of the corresponding code snippet, based on which the aforementioned lexical mismatch issue in a naive keyword-based search engine can be alleviated. A similar idea of improving retrieval by adding extra annotations to items is also explored in document retrieval [47] and image search [61]. However, most of them rely on humans to provide the annotations. Intuitively, our perspective can be interpreted as one type of *machine-machine collaboration*: On the one hand, the NL annotation generated by the code annotation model can serve as a second view of a code snippet (in addition to its programming content) and can be utilized to match with an NL query in code retrieval. On the other hand, with

[1]Code available at https://github.com/LittleYUYU/CoaCor.

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313632

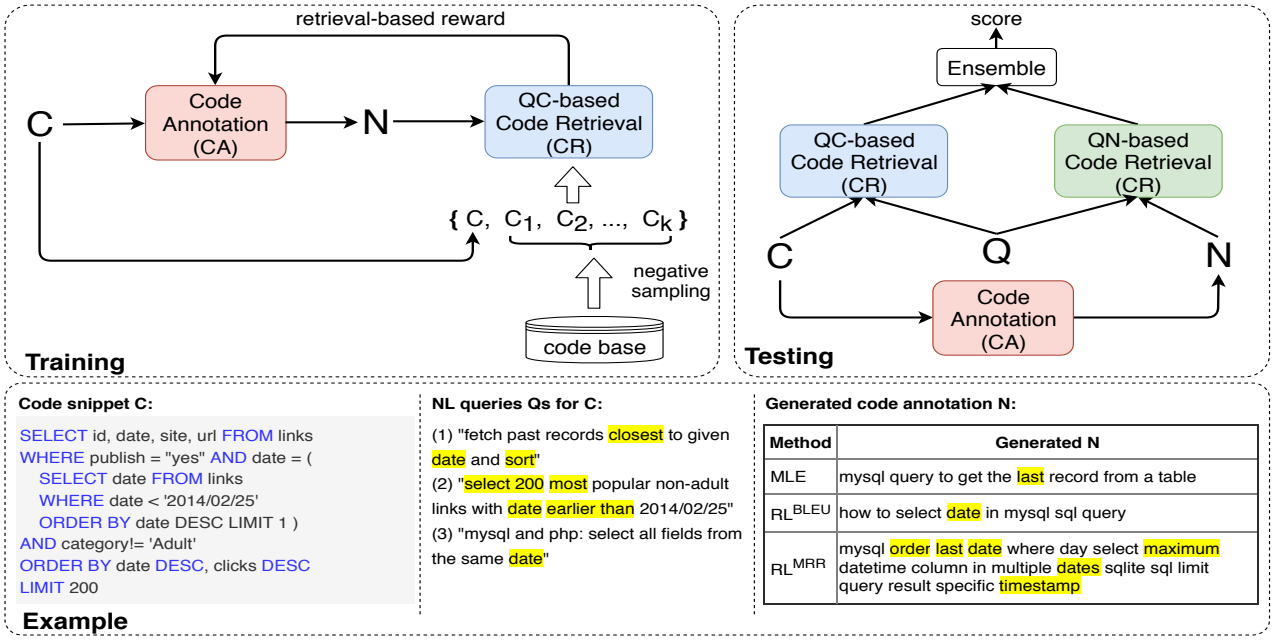


Figure 1: Our CoaCor framework: (1) *Training phase.* A code annotation model is trained via reinforcement learning to maximize retrieval-based rewards given by a QC-based code retrieval model (pre-trained using <NL query, code snippet> pairs). (2) *Testing phase.* Each code snippet is first annotated by the trained CA model. For the code retrieval task, given query Q, a code snippet gets two scores - one matching Q with its code content and the other matching Q with its code annotation N, and is ranked by a simple ensemble strategy[+]. (3) *Example.* We show an example of a code snippet and its associated multiple NL queries in our dataset. The code annotation generated by our framework (denoted as RL^{MRR}) is much more detailed with many keywords *semantically aligned* with Qs, when compared with CA models trained via MLE or RL with BLEU rewards (RL^{BLEU}). [+] We simply use a weighted combination of the two scores, and other ensemble strategies can also apply here.

a goal to facilitate code retrieval, the code annotation model can be stimulated to produce rich and detailed annotations. Unlike existing work [19–21, 58], our goal is *not* to generate an NL annotation as close as possible to a human-provided one; hence, the MLE objective or the BLEU score as rewards for code annotation will not fit our setting. Instead, we design a novel rewarding mechanism in an RL framework, which guides the code annotation model directly based on how effectively the currently generated code annotation distinguishes the code snippet from a candidate set.

Leveraging collaborations and interactions among machine learning models to improve the task performance has been explored in other scenarios. Goodfellow et al. [12] proposed the Generative Adversatial Nets (GANs), where a generative model produces difficult examples to fool a discriminative model and the latter is further trained to conquer the challenge. He et al. [15] proposed another framework called *dual learning*, which jointly learns two dual machine translation tasks (e.g., En -> Fr and Fr -> En). However, none of the existing frameworks are directly applicable to accomplish our goal (i.e., to train a code annotation model for generating annotations that can be utilized for code retrieval).

Figure 1 shows our reinforcement learning-based CoaCor framework. In the training phase, we first train a CR model based on <natural language query, code snippet> (QC) pairs (referred to as QC-based CR model). Then given a code snippet, the CA model generates a sequence of NL tokens as its annotation and receives a reward from the trained CR model, which measures how effectively the generated annotation can distinguish the code snippet from others. We formulate the annotation generation process as a Markov Decision Process [5] and train the CA model to maximize the received reward via an advanced reinforcement learning [53] framework called *Advantage Actor-Critic* or *A2C* [36]. Once the CA model is trained, we use it to generate an NL annotation N for each code snippet C in the code base. Therefore, for each QC pair we originally have, we can derive a QN pair. We utilize the generated annotation as a second view of the code snippet to match with a query and train another CR model based on the derived QN pairs (referred to as QN-based CR model). In the testing phase, given an NL query, we rank code snippets by combining their scores from both the QC-based as well as QN-based CR models, which utilize both the programming content as well as the NL annotation of a code snippet.

On a widely used benchmark dataset [21] and a recently collected large-scale dataset [63], we show that the automatically generated annotation can significantly improve the retrieval performance. More impressively, without looking at the code content, the QN-based CR model trained on our generated code annotations obtains a retrieval performance comparable to one of the state-of-the-art

QC-based CR models. It also surpasses other QN-based CR models trained using code annotations generated by existing CA models.

To summarize, our major contributions are as follows:

- First, we explored a novel perspective of generating useful code annotations *for* code retrieval. Unlike existing work [19, 21, 58], we do not emphasize the n-gram overlap between the generated annotation and the human-provided one. Instead, we examined the real usefulness of the generated annotations and developed a machine-machine collaboration paradigm, where a code annotation model is trained to generate annotations that can be used for code retrieval.
- Second, in order to accomplish our goal, we developed an effective RL-based framework with a novel rewarding mechanism, in which a code retrieval model is directly used to formulate rewards and guide the annotation generation.
- Last, we conducted extensive experiments by comparing our framework with various baselines including state-of-the-art models and variants of our framework. We showed significant improvements of code retrieval performance on both a widely used benchmark dataset and a recently collected large-scale dataset.

The rest of this paper is organized as follows. Section 2 introduces the background on code annotation and code retrieval tasks. Section 3 gives an overview of our proposed framework, with algorithm details followed in Section 4. Experiments are shown in Section 5. Finally, we discuss related work and conclude in Section 6 and 7.

2 BACKGROUND

We adopt the same definitions for code retrieval and code annotation as previous work [9, 21]. Given a natural language query Q and a set of code snippet candidates $\mathbb{C}$, *code retrieval* is to retrieve code snippets $C^* \in \mathbb{C}$ that can match with the query. On the other hand, given a code snippet C, *code annotation* is to generate a natural language (NL) annotation N^* which describes the code snippet appropriately. In this work, we use *code search* and *code retrieval* interchangeably (and same for *code annotation/summary/description*).

Formally, for a training corpus with <natural language query, code snippet> pairs, e.g., those collected from Stack Overflow [51] by [21, 63], we define the two tasks as:

Code Retrieval (CR): Given an NL Query Q, a model F_r will be learnt to retrieve the highest scoring code snippet $C^* \in \mathbb{C}$.

$$C^* = \underset{C \in \mathbb{C}}{\operatorname{argmax}} F_r(Q, C) \tag{1}$$

Code Annotation (CA): For a given code snippet C, the goal is to generate an NL annotation N^* that maximizes a scoring function F_a:

$$N^* = \underset{N}{\operatorname{argmax}} F_a(C, N) \tag{2}$$

Note that one can use the same scoring model for F_r and F_a as in [9, 21], but for most of the prior work [13, 19, 20, 58], which consider either code retrieval or code annotation, researchers usually develop their own models and objective functions for F_r or F_a. In our work, we choose two vanilla models as our base models for CR and CA, but explore a novel perspective of how to train F_a so that it can generate NL annotations that can be used for code retrieval. This perspective is inspired by various machine-machine collaboration mechanisms [15, 28, 55, 59] where one machine learning task can help improve another.

3 FRAMEWORK OVERVIEW

In this section, we first introduce our intuition and give an overview of the entire framework, before diving into more details.

3.1 Intuition behind CoaCor

To the best of our knowledge, previous code annotation work like [19–21, 58] focused on getting a large n-gram overlap between generated and human-provided annotations. However, it is still uncertain (and non-trivial to test) how helpful the generated annotations can be. Driven by this observation, we are the first to examine the real usefulness of the generated code annotations and how they can help a relevant task, of which we choose code retrieval as an example.

Intuitively, CoaCor can be interpreted as a *collaboration mechanism* between code annotation and code retrieval. On the one hand, the annotation produced by the CA model provides a second view of a code snippet (in addition to its programming content) to assist code retrieval. On the other hand, when the CA model is trained to be useful for the retrieval task, we expect it to produce richer and more detailed annotations, which we verify in experiments later.

3.2 Overview

The main challenge to realize the above intuition lies in how to train the CA model effectively. Our key idea to address the challenge is shown in Figure 1.

We first train a base CR model on <natural language query, code snippet> (QC) pairs. Intuitively, a QC-based CR model ranks a code snippet C by measuring how well it matches with the given query Q (in comparison with other code snippets in the code base). From another point of view, a well-trained QC-based CR model can work as a measurement on whether the query Q describes the code snippet C precisely or not. Drawing inspiration from this view, we propose using the trained QC-based CR model to determine whether an annotation describes its code snippet precisely or not and thereby, train the CA model to generate rich annotations to maximize the retrieval-based reward from the CR model. Specifically, given a code snippet C, the CA model generates a sequence of NL words as its annotation N. At the end of the sequence, we let the trained QC-based CR model use N to search for relevant code snippets from the code base. If C can be ranked at top places, the annotation N is treated as well-written and gets a high reward; otherwise, a low reward will be returned. We formulate this generation process as the Markov Decision Process [5] and train the CA model with reinforcement learning [53] (specifically, the *Advantage Actor-Critic* algorithm [36]) to maximize the retrieval-based rewards it can receive from the QC-based CR model. We elaborate the CR and CA model details as well as the RL algorithm for training the CA model in Section 4.1 ∼ 4.3.

Once the CA model is trained, in the testing phase, it generates an NL annotation N for each code snippet C in the code base. Now for each <NL query, code snippet> pair originally in the datasets,

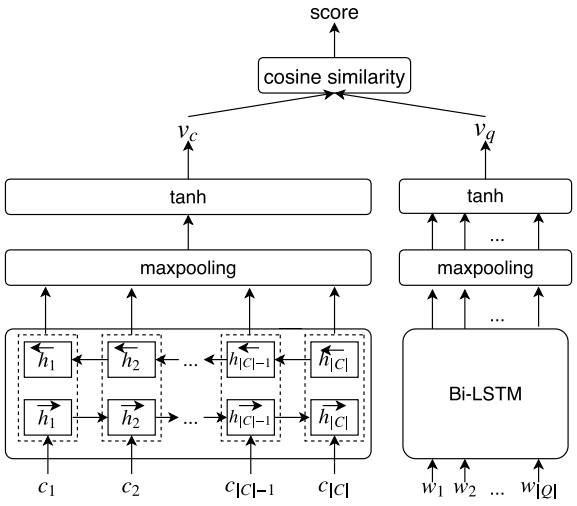


Figure 2: The base code retrieval model encodes the input code snippet C = $(c_1, c_2, ..., c_{|C|})$ and NL query Q = $(w_1, w_2, ..., w_{|Q|})$ into a vector space and outputs a similarity score.

we derive an <NL query, code annotation> (QN) pair and train another CR model based on such QN pairs. This QN-based CR model complements the QC-based CR model, as they respectively use the annotation and programming content of a code snippet to match with the query. We finally combine the matching scores from the two CR models to rank code snippets for a given query.

Note that we aim to outline a general paradigm to explore the perspective of code annotation for code retrieval, where the specific model structures for the two CR models and the CA model can be instantiated in various ways. In this work, we choose one of the simplest and most fundamental deep structures for each of them. Using more complicated model structures will make the training more challenging and we leave it as future work.

4 CoDE ANNOTATION FOR CoDE RETRIEVAL

Now we introduce model and algorithm details in our framework.

4.1 Code Retrieval Model

Both QC-based and QN-based code retrieval models adopt the same deep learning structure as the previous CR work [13]. Here for simplicity, we only illustrate the QC-based CR model in detail, and the QN-based model structure is the same except that we use the generated annotation on the code snippet side.

As shown in Figure 2, given an NL query $Q = w_{1..|Q|}$ and a code snippet $C = c_{1..|C|}$, we first embed the tokens of both code and NL query into vectors through a randomly initialized word embedding matrix, which will be learned during model training. We then use a bidirectional Long Short-Term Memory (LSTM)-based Recurrent Neural Network (RNN) [10, 11, 18, 35, 48, 64] to learn the token representation by summarizing the contextual information from both directions. The LSTM unit is composed of three multiplicative gates. At every time step t, it tracks the state of sequences by controlling how much information is updated into the new hidden state h_t and memory cell g_t from the previous state h_{t-1}, the previous memory cell g_{t-1} and the current input x_t. On the code side, x_t is the embedding vector for c_t, and on the NL query side, it is the embedding vector for w_t. At every time step t, the LSTM hidden state is updated as:

$$
\begin{aligned}
i &= \sigma(\mathbf{W}_i h_{t-1} + \mathbf{U}_i x_t + b_i) \\
f &= \sigma(\mathbf{W}_f h_{t-1} + \mathbf{U}_f x_t + b_f) \\
o &= \sigma(\mathbf{W}_o h_{t-1} + \mathbf{U}_o x_t + b_o) \\
g &= \tanh(\mathbf{W}_g h_{t-1} + \mathbf{U}_g x_t + b_g) \\
g_t &= f \odot g_{t-1} + i \odot g \\
h_t &= o \odot \tanh(g_t)
\end{aligned}
$$

where σ is the element-wise sigmoid function and $\odot$ is the element-wise product. $\mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_g, \mathbf{U}_o$ denote the weight matrices of different gates for input x_t and $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_g$, $\mathbf{W}_o$ are the weight matrices for hidden state h_t, while b_i, b_f, b_g, b_o denote the bias vectors. For simplicity, we denote the above calculation as below (the memory cell vector g_{t-1} is omitted):

$$h_t = \text{LSTM}(x_t, h_{t-1}) \tag{3}$$

The vanilla LSTM's hidden state h_t takes information from the past, knowing nothing about the future. Our CR model instead incorporates a bidirectional LSTM [48] (i.e., Bi-LSTM in Figure 2), which contains a forward LSTM reading a sequence X from start to end and a backward LSTM which reads from end to start. The basic idea of using two directions is to capture past and future information at each step. Then the two hidden states at each time step t are concatenated to form the final hidden state h_t.

$$
\begin{aligned}
\overrightarrow{h_t} &= \text{LSTM}(x_t, \overrightarrow{h_{t-1}}) \\
\overleftarrow{h_t} &= \text{LSTM}(x_t, \overleftarrow{h_{t+1}}) \\
h_t &= [\overrightarrow{h_t}, \overleftarrow{h_t}]
\end{aligned}
$$

Finally, we adopt the commonly used max pooling strategy [24] followed by a *tanh* layer to get the embedding vector for a sequence of length T.

$$v = \tanh(\text{maxpooling}([h_1, h_2, ..., h_T])) \tag{4}$$

By applying the above encoding algorithm, we encode the code snippet C and the NL query Q into v_c and v_q, respectively. Similar to Gu et al. [13], to measure the relevance between the code snippet and the query, we use cosine similarity denoted as $cos(v_q, v_c)$. The higher the similarity, the more related the code is to the query.

Training. The CR model is trained by minimizing a ranking loss similar to [13]. Specifically, for each query Q in the training corpus, we prepare a triple of $<Q, C, C^->$ as a training instance, where C is the correct code snippet that answers Q and C^- is a negative code snippet that does not answer Q (which is randomly sampled from the entire code base). The ranking loss is defined as:

$$\mathcal{L}(\theta) = \sum_{<Q,C,C^->} max(0, \epsilon - cos(v_q, v_c) + cos(v_q, v_{c^-})) \tag{5}$$

where θ denotes the model parameters, ϵ is a constant margin, and v_q, v_c and v_{c^-} are the encoded vectors of Q, C and C^- respectively.

Essentially, the ranking loss is a kind of hinge loss [46] that promotes the cosine similarity between Q and C to be greater than that between Q and C^- by at least a margin ϵ. The training leads the model to project relevant queries and code snippets to be close in the vector space.

4.2 Code Annotation Model

Formally, given a code snippet C, the CA model computes the probability of generating a sequence of NL tokens $n_{1..|N|} = (n_1, n_2, ..., n_{|N|})$ as its annotation N by:

$$P(N|C) = P(n_1|n_0, C) \prod_{t=2}^{|N|} P(n_t|n_{1..t-1}, C) \tag{6}$$

where n_0 is a special token "<START>" indicating the start of the annotation generation, $n_{1..t-1} = (n_1, ..., n_{t-1})$ is the partially generated annotation till time step t-1, and $P(n_t|n_{1..t-1}, C)$ is the probability of producing n_t as the next word given the code snippet C and the generated $n_{1..t-1}$. The generation stops once a special token "<EOS>" is observed.

In this work, we choose the popular sequence-to-sequence model [52] as our CA model structure, which is composed of an encoder and a decoder. We employ the aforementioned bidirectional LSTM-based RNN structure as the encoder for code snippet C, and use another LSTM-based RNN as the decoder to compute Eqn. (6):

$$h_t^{\text{dec}} = \text{LSTM}(n_{t-1}, h_{t-1}^{\text{dec}}), \forall t = 1, ..., |N|$$

where h_t^{dec} is the decoder hidden state at step t, and h_0^{dec} is initialized by concatenating the last hidden states of the code snippet encoder in both directions. In addition, a standard global attention layer [33] is applied in the decoder, in order to attend to important code tokens in C:

$$\tilde{h}_t^{\text{dec}} = \tanh(\mathbf{W}_\alpha[v_{\text{attn}}, h_t^{\text{dec}}])$$

$$v_{\text{attn}} = \sum_{t'=1}^{|C|} \alpha_{t'}\ h_{t'}^{\text{enc}}$$

$$\alpha_{t'} = \text{softmax}((h_{t'}^{\text{enc}})^T h_t^{\text{dec}})$$

where $\alpha_{t'}$ is the attention weight on the t'-th code token when generating the t-th word in the annotation, and $\mathbf{W}_\alpha$ is a learnable weight matrix. Finally, the t-th word is selected based on:

$$P(n_t|n_{0..t-1}, C) = \text{softmax}(\mathbf{W}\tilde{h}_t^{\text{dec}} + b) \tag{7}$$

where $\mathbf{W} \in R^{|V_n| \times d}$, $b \in R^{|V_n|}$ project the d-dim hidden state $\tilde{h}_t^{\text{dec}}$ to the NL vocabulary of size $|V_n|$.

4.3 Training Code Annotation via RL

4.3.1 Code Annotation as Markov Decision Process. Most previous work [9, 19, 21] trained the CA model by maximizing the log-likelihood of generating human annotations, which suffers from two drawbacks: (1) The *exposure bias* issue [4, 44, 45]. That is, during training, the model predicts the next word given the ground-truth annotation prefix, while at testing time, it generates the next word based on previous words generated by itself. This mismatch between training and testing may result in error accumulation in testing phase. (2) More importantly, maximizing the likelihood is not aligned with our goal to produce annotations that can be useful for code retrieval.

To address the above issues, we propose to formulate code annotation within the reinforcement learning (RL) framework [53]. Specifically, the annotation generation process is viewed as a Markov Decision Process (MDP) [5] consisting of four main components:

State. At step t during decoding, a state s_t maintains the source code snippet and the previously generated words $n_{1..t-1}$, i.e., $s_t = \{C, n_{1..t-1}\}$. In particular, the initial decoding state $s_0 = \{C\}$ only contains the given code snippet C. In this work, we take the hidden state vector $\tilde{h}_t^{dec}$ as the vector representation of state s_t, and the MDP is thus processing on a continuous and infinite state space.

Action. The CA model decides the next word (or *action*) $n_t \in V_n$, where V_n is the NL vocabulary. Thus, the action space in our formulation is the NL vocabulary.

Reward. As introduced in Section 3, to encourage it to generate words useful for the code retrieval task, the CA model is rewarded by a well-trained QC-based CR model based on whether the code snippet C can be ranked at the top positions if using the generated *complete* annotation N as a query. Therefore, we define the reward at each step t as:

$$r(s_t, n_t) = \begin{cases} \text{RetrievalReward}(C, n_{1..t}) & \text{if } n_t = \text{<EOS>} \\ 0 & \text{otherwise} \end{cases}$$

where we use the popular ranking metric Mean Reciprocal Rank [57] (defined in Section 5.2) as the RetrievalReward(C, N) value. Note that, in this work, we let the CR model give a valid reward only when the annotation generation stops, and assign a zero reward to intermediate steps. However, other designs such as giving rewards to a partial generation with the reward shaping technique [4] can be reasonable and explored in the future.

Policy. The policy function $P(n_t|s_t)$ takes as input the current state s_t and outputs the probability of generating n_t as the next word. Given our definition about state s_t and Eqn. (7),

$$P(n_t|s_t) = P(n_t|n_{1..t-1}, C) = \text{softmax}(\mathbf{W}\tilde{h}_t^{dec} + b) \tag{8}$$

Here, the policy function is stochastic in that the next word can be sampled according to the probability distribution, which allows action space exploration and can be optimized using policy gradient methods [53].

The objective of the CA model training is to find a policy function $P(N|C)$ that maximizes the expected accumulative future reward:

$$\max_\phi \mathcal{L}(\phi) = \max_\phi \mathbb{E}_{N \sim P(\cdot|C;\phi)}[R(C, N)] \tag{9}$$

where ϕ is the parameter of P and $R(C, N) = \sum_{t=1}^{|N|} r(s_t, n_t)$ is the accumulative future reward (called "return").

The gradient of the above objective is derived as below.

$$\begin{aligned} \nabla_\phi \mathcal{L}(\phi) &= \mathbb{E}_{N \sim P(\cdot|C;\phi)}[R(C, N) \nabla_\phi \log P(N|C;\phi)] \\ &= \mathbb{E}_{n_{1..|N|} \sim P(\cdot|C;\phi)}[\sum_{t=1}^{|N|} R_t(s_t, n_t) \nabla_\phi \log P(n_t|n_{1..t-1}, C;\phi)] \end{aligned} \tag{10}$$

where $R_t(s_t, n_t) = \sum_{t' \geq t} r(s_{t'}, n_{t'})$ is the return for generating word n_t given state s_t.

4.3.2 Advantage Actor-Critic for Code Annotation. Given the gradient in Eqn. (10), the objective in Eqn. (9) can be optimized by policy gradient approaches such as REINFORCE [62] and Q-value Actor-Critic algorithm [4, 54]. However, such methods may yield very high variance when the action space (i.e., the NL vocabulary) is large and suffer from biases when estimating the return of rarely taken actions [39], leading to unstable training. To tackle the challenge, we resort to the more advanced *Advantage Actor-Critic* or *A2C* algorithm [36], which has been adopted in other sequence generation tasks [39, 58]. Specifically, the gradient function in Eqn. (10) is replaced by an *advantage* function:

$$\nabla_\phi \mathcal{L}(\phi) = \mathbb{E}[\sum_{t=1}^{|N|} A_t \nabla_\phi \log P(n_t|n_{1..t-1}, C; \phi)] \tag{11}$$
$$A_t = R_t(s_t, n_t) - V(s_t)$$
$$V(s_t) = \mathbb{E}_{\hat{n}_t \sim P(\cdot|n_{1..t-1}, C)}[R_t(s_t, \hat{n}_t)]$$

where $V(s_t)$ is the state value function that estimates the future reward given the current state s_t. Intuitively, $V(s_t)$ works as a *baseline* function [62] to help the model assess its action n_t more precisely: When advantage A_t is greater than zero, it means that the return for taking action n_t is better than the "average" return over all possible actions, given state s_t; otherwise, action n_t performs worse than the average.

Following previous work [36, 39, 58], we approximate $V(s_t)$ by learning another model $V(s_t; \rho)$ parameterized by ρ, and the RL framework thus contains two components: the policy function $P(N|C; \phi)$ that generates the annotation (called "Actor"), and the state value function $V(s_t; \rho)$ that approximates the return under state s_t (called "Critic"). Similar to the actor model (i.e., the CA model in Section 4.2), we train a separate attention-based sequence-to-sequence model as the critic network. The critic value is finally computed by:

$$V(s_t; \rho) = \mathbf{w}_\rho^T (\tilde{h}_t^{dec})_{crt} + b_\rho \tag{12}$$

where $(\tilde{h}_t^{dec})_{crt} \in R^d$ is the critic decoder hidden state at step t, and $\mathbf{w}_\rho \in R^d$, $b_\rho \in R$ are trainable parameters that project the hidden state to a scalar value (i.e., the estimated state value).

The critic network is trained to minimize the Mean Square Error between its estimation and the true state value:

$$\min_\rho \mathcal{L}(\rho) = \min_\rho \mathbb{E}_{N \sim P(\cdot|C)}[\sum_{t=1}^{|N|} (V(s_t; \rho) - R(C, N))^2] \tag{13}$$

The entire training procedure of CoaCor is shown in Algorithm 1.

4.4 Generated Annotations for Code Retrieval

As previously shown in Figure 1, in the testing phase, we utilize the generated annotations to assist the code retrieval task. Now we detail the procedure in Algorithm 2. Specifically, for each <NL query, code snippet> pair (i.e., QC pair) in the dataset, we first derive an <NL query, code annotation> pair (i.e., QN pair) using the code annotation model. We then build another code retrieval (CR) model based on the QN pairs in our training corpus. In this work, for simplicity, we choose the same structure as the QC-based CR model (Section 4.1) to match QN pairs. However, more advanced methods for modeling the semantic similarity between two NL sentences (e.g., previous work on NL paraphase detection [16, 26]) are applicable and can be explored as future work.

Algorithm 1 : Training Procedure for CoaCor.

Input: <NL query, code snippet> (QC) pairs in training set, number of iterations E.

1: Train a base code retrieval model based on QC pairs, according to Eqn. (5).
2: Initialize a base code annotation model (ϕ) and pretrain it via MLE according to Eqn. (7), using Q as the desired N for C.
3: Pretrain a critic network (ρ) according to Eqn. (13).
4: **for** $iteration = 1$ to E **do**
5: Receive a code snippet C.
6: Sample an annotation $N \sim P(\cdot|C; \phi)$ according to Eqn. (8).
7: Receive the final reward $R(C, N)$.
8: Update the code annotation model (ϕ) using Eqn. (11).
9: Update the critic network (ρ) using Eqn. (13).
10: **end for**

Algorithm 2 : Generated Annotations for Code Retrieval.

Input: NL query Q, code snippet candidate C.
Output: The matching score, score(Q, C).

1: Receive score$_1(Q, C) = cos(v_q, v_c)$ from a QC-based code retrieval model.
2: Generate a code annotation $N \sim P(\cdot|C; \phi)$ via greedy search, according to Eqn. (8).
3: Receive score$_2(Q, C) = cos(v_q, v_n)$ from a QN-based code retrieval model.
4: Calculate score(Q, C) according to Eqn. (14).

The final matching score between query Q and code snippet C combines those from the QN-based and QC-based CR model:

$$\text{score}(Q, C) = \lambda * cos(v_q, v_n) + (1 - \lambda) * cos(v_q, v_c) \tag{14}$$

where v_q, v_c, v_n are the encoded vectors of Q, C, and the code annotation N respectively. $\lambda \in [0, 1]$ is a weighting factor for the two scores to be tuned on the validation set.

5 EXPERIMENTS

In this section, we conduct extensive experiments and compare our framework with various models to show its effectiveness.

5.1 Experimental Setup

Dataset. (1) We experimented with the **StaQC** dataset presented by Yao et al. [63]. The dataset contains 119,519 SQL <question title, code snippet> pairs mined from Stack Overflow [51], making itself the largest-to-date in SQL domain. In our code retrieval task, the question title is considered as the NL query Q, which is paired with the code snippet C to form the QC pair. We randomly selected 75% of the pairs for training, 10% for validation (containing 11,900 pairs), and the left 15% for testing (containing 17,850 pairs). As mentioned in [63], the dataset may contain multiple code snippets for the same NL query. We examine the dataset split to ensure that alternative relevant code snippets for the same query would not be sampled as negative code snippets when training the CR model. For pretraining the CA model, we consider the question title as a

code annotation N and form QN pairs accordingly. **(2)** Iyer et al. [21] collected two small sets of SQL code snippets (called "**DEV**" and "**EVAL**" respectively) from Stack Overflow for validation and testing. In addition to the originally paired question title, each code snippet is manually annotated by two different NL descriptions. Therefore, in total, each set contains around 100 code snippets with three NL descriptions (resulting in around 300 QC pairs). We use them as additional datasets for model comparison.[2] Following [21], QC pairs occuring in DEV and EVAL set or being used as negative code snippets by them are removed from the StaQC training set.

Data Preprocessing. We followed [21] to perform code tokenization, which replaced table/column names with placeholder tokens and numbered them to preserve their dependencies. For text tokenization, we utilized the "word_tokenize" tool in the NLTK toolkit [7]. All code tokens and NL tokens with a frequency of less than 2 were replaced with an <UNK> token, resulting in totally 7726 code tokens and 7775 word tokens in the vocabulary. The average lengths of the NL query and the code snippet are 9 and 60 respectively.

5.2 Evaluation

We evaluate a model's retrieval performance on four datasets: the validation (denoted as "StaQC-val") and test set (denoted as "StaQC-test") from StaQC [63], and the DEV and EVAL set from [21]. For each <NL query Q, code snippet C> pair (or QC pair) in a dataset, we take C as a positive code snippet and randomly sample K negative code snippets from all others except C in the dataset,[3] and calculate the rank of C among the $K+1$ candidates. We follow [9, 21, 63] to set $K = 49$. The retrieval performance of a model is then assessed by the Mean Reciprocal Rank (MRR) metric [57] over the entire set $\mathcal{D} = \{(Q_1, C_1), (Q_2, C_2), ..., (Q_{|\mathcal{D}|}, C_{|\mathcal{D}|})\}$:

$$MRR = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \frac{1}{Rank_i}$$

where $Rank_i$ is the rank of C_i for query Q_i. The higher the MRR value, the better the code retrieval performance.

5.3 Methods to Compare

In order to test the effectiveness of our CoaCor framework, we compare it with both existing baselines and our proposed variants.

Existing Baselines. We choose the following state-of-the-art code retrieval models, which are based on QC pairs, for comparison.

- Deep Code Search (DCS) [13]. The original DCS model [13] adopts a similar structure as Figure 2 for CR in Java domain. To learn the vector representations for code snippets, in addition to code tokens, it also considers features like function names and API sequences, all of which are combined into a fully connected layer. In our dataset, we do not have these features, and thus slightly modify their original model to be the same as our QC-based CR model (Figure 2).
- CODE-NN [21]. CODE-NN is one of the state-of-the-art models for both code retrieval and code annotation. Its core component is an LSTM-based RNN with an attention mechanism, which models the probability of generating an NL sentence conditioned on a given code snippet. For code retrieval, given an NL query, CODE-NN computes the likelihood of generating the query as an annotation for each code snippet and ranks code snippets based on the likelihood.

QN-based CR Variants. As discussed in Section 4.4, a trained CA model is used to annotate each code snippet C in our datasets with an annotation N. The resulting <NL query Q, code annotation N> pairs can be used to train a QN-based CR model. Depending on how we train the CA model, we have the following variants:

- QN-MLE. Similar to most previous work [9, 19, 21], we simply train the CA model in the standard MLE manner, i.e., by maximizing the likelihood of a human-provided annotation.
- QN-RL$^{\text{BLEU}}$. As introduced in Section 1, Wan et al. [58] proposed to train the CA model via reinforcement learning with BLEU scores [41] as rewards. We compare this variant with our rewarding mechanism.
- QN-RL$^{\text{MRR}}$. In our CoaCor framework, we propose to train the CA model using retrieval rewards from a QC-based CR model (see Section 3). Here we use the MRR score as the retrieval reward.

Since CODE-NN [21] can be used for code annotation as well, we also use its generated code annotations to train a QN-based CR model, denoted as "QN-CodeNN".[4]

Ensemble CR Variants. As introduced in Section 4.4, we tried ensembling the QC-based CR model and the QN-based CR model to improve the retrieval performance. We choose the DCS structure as the QC-based CR model as mentioned in Section 4.1. Since different QN-based CR models can be applied, we present the following 4 variants: (1) QN-MLE + DCS, (2) QN-RL$^{\text{BLEU}}$ + DCS, (3) QN-RL$^{\text{MRR}}$ + DCS, and (4) QN-CodeNN + DCS, where QN-MLE, QN-RL$^{\text{BLEU}}$, QN-RL$^{\text{MRR}}$, and QN-CodeNN have been introduced.

5.4 Implementation Details

Our implementation is based on Pytorch [42]. For CR models, we set the embedding size of words and code tokens to 200, and chose batch size in {128, 256, 512}, LSTM unit size in {100, 200, 400} and dropout rate [50] in {0.1, 0.35, 0.5}. A small, fixed ϵ value of 0.05 is used in all the experiments. Hyper-parameters for each model were chosen based on the DEV set. For CODE-NN baseline, we followed Yao et al. [63] to use the same model hyper-parameters as the original paper, except that the dropout rate is tuned in {0.5, 0.7}. The StaQC-val set was used to decay the learning rate and the best model parameters were decided based on the retrieval performance on the DEV set.

For CA models, the embedding size of words and code tokens and the LSTM unit size were selected from {256, 512}. The dropout rate is selected from {0.1, 0.3, 0.5} and the batch size is 64. We updated model parameters using the Adam optimizer [25] with learning rate 0.001 for MLE training and 0.0001 for RL training. The maximum length of the generated annotation is set to 20. For CodeNN, MLE-based and RL$^{\text{BLEU}}$-based CA models, the best model parameters

[2] Previous work [9, 21] used only one of the three descriptions while we utilize all of them to enrich and enlarge the datasets for a more reliable evaluation.

[3] For DEV and EVAL, we use the same negative examples as [21].

[4] There is another recent code annotation method named DeepCom [19]. We did not include it as baseline, since it achieved a similar performance as our MLE-based CA model (see Table 3) when evaluated with the standard BLEU script by [21].

Model	DEV	EVAL	StaQC-val	StaQC-test
Existing (QC-based) CR Baselines				
DCS [13]	0.566	0.555	0.534	0.529
CODE-NN [21]	0.530	0.514	0.526	0.522
QN-based CR Variants				
QN-CodeNN	0.369	0.360	0.336	0.333
QN-MLE	0.429	0.411	0.427	0.424
QN-RL^{BLEU}	0.426	0.402	0.386	0.381
QN-RL^{MRR} (ours)	0.534	0.512	0.516	0.523
Ensemble CR Variants				
QN-CodeNN + DCS	0.566	0.555	0.534	0.529
QN-MLE + DCS	0.571	0.561	0.543	0.537
QN-RL^{BLEU} + DCS	0.570	0.559	0.541	0.534
QN-RL^{MRR} + DCS (ours)	0.582*	**0.572***	0.558*	0.559*
QN-RL^{MRR} + CODE-NN (ours)	**0.586***	0.571*	**0.575***	**0.576***

Table 1: The main code retrieval results (MRR). * denotes significantly different from DCS [13] in one-tailed t-test ($p < 0.01$).

were picked based on the model's BLEU score on DEV, while for RL^{MRR}-based CA model, we chose the best model according to its MRR reward on StaQC-val. For RL models, after pretraining the actor network via MLE, we first pretrain the critic network for 10 epochs, then jointly train the two networks for 40 epochs. Finally, for ensemble variants, the ensemble weight λ in all variants is selected from $0.0 \sim 1.0$ based on its performance on DEV.

5.5 Results

To understand our CoaCor framework, we first show several concrete examples to understand the differences between annotations generated by our model and by baseline/variant models, and then focus on two research questions (RQs):

- **RQ1 (CR improves CA)**: Is the proposed retrieval reward-driven CA model capable of generating rich code annotations that can be used for code retrieval (i.e., can represent the code snippet and distinguish it from others)?
- **RQ2 (CA improves CR)**: Can the generated annotations further improve existing QC-based code retrieval models?

5.5.1 Qualitative Analysis. Table 2 presents two examples of annotations generated by each CA model. Note that we do not target at human language-like annotations; rather, we focus on annotations that can describe/capture the functionality of a code snippet. In comparison with baseline CA models, our proposed RL^{MRR}-based CA model can produce more concrete and precise descriptions for corresponding code snippets. As shown in Example 1, the annotation generated by RL^{MRR} covers more conceptual keywords semantically aligned with the three NL queries (e.g., "average", "difference", "group"), while the baseline CODE-NN and the variants generate short descriptions covering a very limited amount of conceptual keywords (e.g., without mentioning the concept "subtracting").

We also notice that our CA model can generate different forms of a stem word (e.g., "average", "avg" in Example 1), partly because the retrieval-based reward tends to make the generated annotation semantically aligned with the code snippet and these diverse forms of words can help strengthen such semantic alignment and benefit the code retrieval task when there are various ways to express user search intent.

5.5.2 Code Retrieval Performance Evaluation. Table 1 shows the code retrieval evaluation results, based on which we discuss **RQ1** and **RQ2** as below:

RQ1: To examine whether or not the code annotations generated by a CA model can represent the corresponding code snippet in the code retrieval task, we analyze its corresponding QN-based CR model, which retrieves relevant code snippets by matching the NL query Q with the code annotation N generated by this CA model. Across all of the four datasets, our proposed QN-RL^{MRR} model, which is based on a retrieval reward-driven CA model, achieves the best results and outperforms other QN-based CR models by a wide margin of around $0.1 \sim 0.2$ absolute MRR. More impressively, its performance is already on a par with the CODE-NN model, which is one of the state-of-the-art models for the code retrieval task, even though it understands a code snippet solely based on its annotation and without looking at the code content. This demonstrates that the code annotation generated by our proposed framework can reflect the semantic meaning of each code snippet more precisely.

To further understand whether or not the retrieval-based reward can serve as a better reward metric than BLEU (in terms of stimulating a CA model to generate useful annotations), we present the BLEU score of each CA model in Table 3.[5] When connecting this table with Table 1, we observe an inverse trend: For the RL^{BLEU} model which is *trained for* a higher BLEU score, although it can improve the MLE-based CA model by more than 2% absolute BLEU on three sets, it harms the latter's ability on producing useful code annotations (as revealed by the performance of QN-RL^{BLEU} in Table 1, which is worse than QN-MLE by around 0.04 absolute MRR on StaQC-val and StaQC-test). In contrast, our proposed RL^{MRR} model, despite getting the lowest BLEU score, is capable of generating annotations useful for the retrieval task. This is mainly because that

[5]BLEU is evaluated with the script provided by Iyer et al. [21]: https://github.com/sriniiyer/codenn/blob/master/src/utils/bleu.py.

Model	Annotation
Example 1 from EVAL set	
SQL Code	SELECT col3, Format(Avg([col2]-[col1]),"hh:mm:ss") AS TimeDiff FROM Table1 GROUP BY col3;
Human-provided	(1) find the average time in hours , mins and seconds between 2 values and show them in groups of another column (2) group rows of a table and find average difference between them as a formatted date (3) ms access average after subtracting
CODE-NN	how do i get the average of a column in sql?
MLE	how to get average of the average of a column in sql
RL^{BLEU}	how to average in sql query
RL^{MRR}	average avg calculating difference day in access select distinct column value sql group by month mysql format date function?
Example 2 from StaQC-test set	
SQL Code	SELECT Group_concat(DISTINCT(p.products_id)) AS comma_separated, COUNT(DISTINCT p.products_id) AS product_count FROM ...
Human-provided	how to count how many comma separated values in a group_concat
CODE-NN	how do i get the count of distinct rows?
MLE	mysql query to get count of distinct values in a column
RL^{BLEU}	how to count in mysql sql query
RL^{MRR}	group_concat count concatenate distinct comma group mysql concat column in one row rows select multiple columns of same id result

Table 2: Two examples of code snippets and their annotations generated by different CA models. "Human-provided" refers to (multiple) human-provided NL annotations or queries. Words semantically aligned between the generated and the human-provided annotations are highlighted.

BLEU score calculates surface form overlaps while the retrieval-based reward measures the semantically aligned correspondences.

These observations imply an interesting conclusion: *Compared with BLEU, a (task-oriented) semantic measuring reward, such as our retrieval-based MRR score, can better stimulate the model to produce detailed and useful generations.* This is in line with the recent discussions on whether the automatic BLEU score is an appropriate evaluation metric for generation tasks or not [30, 40]. In our work, we study the potential to use the performance of a relevant model to guide the learning of the target model, which can be generalized to many other scenarios, e.g., conversation generation [27], machine translation [4, 44], etc.

RQ2: We first inspect whether the generated code annotations can assist the base code retrieval model (i.e., DCS) or not by comparing several ensemble CR variants. It is shown that, by simply combining the matching scores from QN-RL^{MRR} and DCS with a weighting factor, our proposed model is able to significantly outperform the DCS model by 0.01 ~ 0.03 and the CODE-NN baseline by 0.03 ~ 0.06 consistently across all datasets, showing the advantage of utilizing code annotations for code retrieval. Particularly, the best performance is achieved when the ensemble weight $\lambda = 0.4$ (i.e., 0.4 weight on the QN-based CR score and 0.6 on the QC-based CR score), meaning that the model relies heavily on the code annotation to achieve better performance.

Model	DEV	EVAL	StaQC-val	StaQC-test
CODE-NN [21]	17.43	16.73	8.89	8.96
MLE	18.99	19.87	10.52	10.55
RL^{BLEU}	21.12	18.52	12.72	12.78
RL^{MRR}	8.09	8.52	5.56	5.60

Table 3: The BLEU score of each code annotation model.

In contrast, QN-CodeNN, QN-MLE and QN-RL^{BLEU} can hardly improve the base DCS model, and their best performances are all achieved when the ensemble weight $\lambda = 0.0 \sim 0.2$, indicating little help from annotations generated by CODE-NN, MLE-based and BLEU-rewarded CA. This is consistent with our conclusions to RQ1.

We also investigate the benefit of our generated annotations to other code retrieval models (besides DCS) by examining a baseline "QN-RL^{MRR} + CODE-NN", which combines QN-RL^{MRR} and CODE-NN (as a QC-based CR model) to score a code snippet candidate. As mentioned in Section 5.3, CODE-NN scores a code snippet by the likelihood of generating the given NL query when taking this code snippet as the input. Since the score is in a different range from the cosine similarity given by QN-RL^{MRR}, we first rescale it by taking its log value and dividing it by the largest absolute log score among all code candidates. The rescaled score is then combined with the cosine similarity score from QN-RL^{MRR} following Eqn. (14). The result is shown in the last row of Table 1. It is very impressive that, with the help of QN-RL^{MRR}, the CODE-NN model can be improved by ≥ 0.05 absolute MRR value across all test sets.

In summary, through extensive experiments, we show that our proposed framework can generate code annotations that are much more useful for building effective code retrieval models, in comparison with existing CA models or those trained by MLE or BLEU-based RL. Additionally, the generated code annotations can further improve the retrieval performance, when combined with existing CR models like DCS and CODE-NN.

6 DISCUSSION

In this work, we propose a novel perspective of using a relevant downstream task (i.e., code retrieval) to guide the learning of a target task (i.e., code annotation), illustrating a novel machine-machine collaboration paradigm. It is shown that the annotations generated by the RL^{MRR} CA model (trained with rewards from the DCS model) can boost the performance of the CODE-NN model, which was not involved in any stage of the training process. It is interesting to explore more about machine-machine collaboration mechanisms, where multiple models for either the same task or relevant tasks can be utilized in tandem to provide different views or effective rewards to improve the final performance.

In terms of training, we also experimented with directly using a QN-based CR model or an ensemble CR model for rewarding the

CA model. However, these approaches do not work well, since we do not have a rich set of QN pairs as training data in the beginning. Collecting paraphrases of queries to form QN pairs is non-trivial, which we leave to the future.

Finally, our CoaCor framework is applicable to other programming languages, such as Python and C#, extension to which is interesting to study as future work.

7 RELATED WORK

Code Retrieval. As introduced in Section 1, code retrieval has been studied widely with information retrieval methods [14, 17, 23, 32, 56] and recent deep learning models [3, 13, 21]. Particularly, Keivanloo et al. [23] extracted abstract programming patterns and their associated NL keywords from code snippets in a code base, with which a given NL query can be projected to a set of associated programming patterns facilitating code content-based search. Similarly, Vinayakarao et al. [56] built an entity discovery system to mine NL phrases and their associated syntactic patterns, based on which they annotated each line of code snippets with NL phrases. Such annotations were utilized to improve NL keyword-based search engines. Different from these work, we construct a neural network-based code annotation model to describe the functionality of an entire code snippet. Our code annotation model is explicitly trained to produce meaningful words that can be used for code search. In our framework, the code retrieval model adopts a similar deep structure as the Deep Code Search model proposed by Gu et al. [13], which projects a NL query and a code snippet into a vector space and measures the cosine similarity between them.

Code Annotation. Code annotation/summarization has drawn a lot of attention in recent years. Earlier works tackled the problem using template-based approaches [37, 49] and topic n-grams models [38] while the recent techniques [2, 19–22, 31] are mostly built upon deep neural networks. Specifically, Sridhara et al. [49] developed a software word usage model to identify action, theme and other arguments from a given code snippet, and generated code comments with templates. Allamanis et al. [2] employed convolution on the input tokens to detect local time-invariant and long-range topical attention features to summarize a code snippet into a short, descriptive function name-like summary. Most related to our work are [19] and [58], which utilized sequence-to-sequence networks with attention over code tokens to generate natural language annotations. They aimed to generate NL annotations as close as possible to human-provided annotations for human readability, and hence adopted the Maximum Likelihood Estimation (MLE) or BLEU score optimization as the objective. However, our goal is to generate code annotations *which can be used for code retrieval*, and therefore we design a retrieval-based reward to drive our training.

Deep Reinforcement Learning for Sequence Generation. Reinforcement learning (RL) [53] has shown great success in various tasks where an agent has to perform multiple actions before obtaining a reward or when the metric to optimize is not differentiable. The sequence generation tasks, such as machine translation [4, 39, 44], image captioning [45], dialogue generation [27] and text summarization [43], have all benefitted from RL to address the *exposure bias* issue [4, 44, 45] and to directly optimize the model towards a certain metric (e.g., BLEU). Particularly, Ranzato et al. [44] were among the first to successfully apply the REINFORCE algorithm [62] to train RNN models for several sequence generation tasks, indicating that directly optimizing the metric used at test time can lead to significantly better models than those trained via MLE. Bahdanau et al. [4] additionally learned a *critic* network to better estimate the return (i.e., future rewards) of taking a certain action under a specific state, and trained the entire generation model via the *Actor-Critic* algorithm [54]. We follow Nguyen et al. [39] and Wan et al. [58] to further introduce an *advantage* function and train the code annotation model via the *Advantage Actor-Critic* algorithm [36], which is helpful for reducing biases from rarely taken actions. However, unlike their work, the reward in our framework is based on the performance on a different yet relevant task (i.e., code retrieval), rather than the BLEU metric.

Machine-Machine Collaboration via Adversarial Training and Dual/Joint Learning. Various kinds of machine-machine collaboration mechanisms have been studied in many scenarios [12, 15, 28, 55, 59]. For example, Goodfellow et al. [12] proposed the Generative Adversarial Nets (GANs) framework, where a generative model generates images to fool a discriminative classifier, and the latter is further improved to distinguish the generated from the real ones. He et al. [15] proposed the dual learning framework and jointly optimized the machine translation from English to French and from French to English. Li et al. [29] trained a paraphrase generator by rewards from a paraphrase evaluator model. In the context of code retrieval and annotation, Chen and Zhou [9] and Iyer et al. [21] showed that their models can be used directly or with slight modification for both tasks, but their training objective only considered one of the two tasks. All these frameworks are not directly applicable to achieve our goal, i.e., training a code annotation model to generate rich NL annotations that can be used for code search.

8 CONCLUSION

This paper explored a novel perspective of generating code annotations *for* code retrieval. To this end, we proposed a reinforcement learning-based framework (named "CoaCor") to maximize a retrieval-based reward. Through comprehensive experiments, we demonstrated that the annotation generated by our framework is more detailed to represent the semantic meaning of a code snippet. Such annotations can also improve the existing code content-based retrieval models significantly. In the future, we will explore other usages of the generated code annotations, as well as generalizing our framework to other tasks such as machine translation.

ACKNOWLEDGMENTS

This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, Fujitsu gift grant, and Ohio Supercomputer Center [8]. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

REFERENCES

[1] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 81.

[2] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In *International Conference on Machine Learning*. 2091–2100.

[3] Miltos Allamanis, Daniel Tarlow, Andrew Gordon, and Yi Wei. 2015. Bimodal modelling of source code and natural language. In *International Conference on Machine Learning*. 2123–2132.

[4] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086* (2016).

[5] Richard Bellman. 1957. A Markovian decision process. *Journal of Mathematics and Mechanics* (1957), 679–684.

[6] Ted J Biggerstaff, Bharat G Mitbander, and Dallas E Webster. 1994. Program understanding and the concept assignment problem. *Commun. ACM* 37, 5 (1994), 72–82.

[7] Steven Bird and Edward Loper. 2004. NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 31.

[8] Ohio Supercomputer Center. 1987. Ohio Supercomputer Center. http://osc.edu/ark:/19495/f5s1ph73.

[9] Qingying Chen and Minghui Zhou. 2018. A neural framework for retrieval and summarization of source code. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 826–831.

[10] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).

[11] Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, Vol. 1. IEEE, 347–352.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[13] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, 933–944.

[14] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 842–851.

[15] Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tieyan Liu, and Wei-Ying Ma. 2016. Dual learning for machine translation. In *Advances in Neural Information Processing Systems*. 820–828.

[16] Hua He and Jimmy Lin. 2016. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 937–948.

[17] Emily Hill, Manuel Roldan-Vega, Jerry Alan Fails, and Greg Mallet. 2014. NL-based query refinement and contextualized code search results: A user study. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 34–43.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[19] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep Code Comment Generation. In *Proceedings of the 2017 26th IEEE/ACM International Conference on Program Comprehension*. ACM.

[20] Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. 2018. Summarizing Source Code with Transferred API Knowledge. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 2269–2275. https://doi.org/10.24963/ijcai.2018/314

[21] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 2073–2083.

[22] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 135–146.

[23] Iman Keivanloo, Juergen Rilling, and Ying Zou. 2014. Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 664–675.

[24] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[25] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[26] Wuwei Lan and Wei Xu. 2018. Neural Network Models for Paraphrase Identification, Semantic Textual Similarity, Natural Language Inference, and Question Answering. *arXiv preprint arXiv:1806.04330* (2018).

[27] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541* (2016).

[28] Yikang Li, Nan Duan, Bolei Zhou, Xiao Chu, Wanli Ouyang, Xiaogang Wang, and Ming Zhou. 2018. Visual question generation as dual task of visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6116–6124.

[29] Zichao Li, Xin Jiang, Lifeng Shang, and Hang Li. 2018. Paraphrase Generation with Deep Reinforcement Learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 3865–3878.

[30] Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023* (2016).

[31] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A neural architecture for generating natural language descriptions from source code changes. *arXiv preprint arXiv:1704.04856* (2017).

[32] Meili Lu, Xiaobing Sun, Shaowei Wang, David Lo, and Yucong Duan. 2015. Query expansion via wordnet for effective code search. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 545–549.

[33] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[34] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Chen Fu, and Qing Xie. 2012. Exemplar: A source code search engine for finding highly relevant applications. *IEEE Transactions on Software Engineering* 38, 5 (2012), 1069–1087.

[35] Larry Medsker and Lakhmi C Jain. 1999. *Recurrent neural networks: design and applications*. CRC press.

[36] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[37] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K Vijay-Shanker. 2013. Automatic generation of natural language summaries for java classes. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*. IEEE, 23–32.

[38] Dana Movshovitz-Attias and William W Cohen. 2013. Natural language models for predicting programming comments. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Vol. 2. 35–40.

[39] Khanh Nguyen, Hal Daumé III, and Jordan Boyd-Graber. 2017. Reinforcement Learning for Bandit Neural Machine Translation with Simulated Human Feedback. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 1464–1474.

[40] Jekaterina Novikova, Ondřej Dušek, Amanda Cercas Curry, and Verena Rieser. 2017. Why we need new evaluation metrics for nlg. *arXiv preprint arXiv:1707.06875* (2017).

[41] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 311–318.

[42] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[43] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304* (2017).

[44] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732* (2015).

[45] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *CVPR*, Vol. 1. 3.

[46] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. 2004. Are loss functions all the same? *Neural Computation* 16, 5 (2004), 1063–1076.

[47] Falk Scholer, Hugh E Williams, and Andrew Turpin. 2004. Query association surrogates for web search. *Journal of the American Society for Information Science and Technology* 55, 7 (2004), 637–650.

[48] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[49] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K Vijay-Shanker. 2010. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 43–52.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[51] Stack Overflow. 2018. Stack Overflow. https://stackoverflow.com/.

[52] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[53] Richard S Sutton and Andrew G Barto. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

[54] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[55] Duyu Tang, Nan Duan, Tao Qin, Zhao Yan, and Ming Zhou. 2017. Question answering and question generation as dual tasks. *arXiv preprint arXiv:1706.02027* (2017).

[56] Venkatesh Vinayakarao, Anita Sarma, Rahul Purandare, Shuktika Jain, and Saumya Jain. 2017. ANNE: Improving Source Code Search Using Entity Retrieval Approach. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. ACM, New York, NY, USA, 211–220. https://doi.org/10.1145/3018661.3018691

[57] Ellen M Voorhees et al. 1999. The TREC-8 Question Answering Track Report.. In *Trec*, Vol. 99. 77–82.

[58] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 397–407.

[59] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 515–524.

[60] Xiaoran Wang, Yifan Peng, and Benwen Zhang. 2018. Comment Generation for Source Code: State of the Art, Challenges and Opportunities. *arXiv preprint arXiv:1802.02971* (2018).

[61] Wikipedia contributors. 2019. Google Image Labeler — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Google_Image_Labeler&oldid=881738511 [Online; accessed 4-February-2019].

[62] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[63] Ziyu Yao, Daniel S Weld, Wei-Peng Chen, and Huan Sun. 2018. StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow. *arXiv preprint arXiv:1803.09371* (2018).

[64] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* (2014).

Leveraging *2-hop* Distant Supervision from Table Entity Pairs for Relation Extraction

Xiang Deng
The Ohio State University
deng.595@buckeyemail.osu.edu

Huan Sun
The Ohio State University
sun.397@osu.edu

Abstract

Distant supervision (DS) has been widely used to automatically construct (noisy) labeled data for relation extraction (RE). Given two entities, distant supervision exploits sentences that directly mention them for predicting their semantic relation. We refer to this strategy as 1-hop DS, which unfortunately may not work well for long-tail entities with few supporting sentences. In this paper, we introduce a new strategy named *2-hop* DS to enhance distantly supervised RE, based on the observation that there exist a large number of relational tables on the Web which contain entity pairs that share common relations. We refer to such entity pairs as *anchors* for each other, and collect all sentences that mention the anchor entity pairs of a given target entity pair to help relation prediction. We develop a new neural RE method REDS2 in the multi-instance learning paradigm, which adopts a hierarchical model structure to fuse information respectively from 1-hop DS and 2-hop DS. Extensive experimental results on a benchmark dataset show that REDS2 can consistently outperform various baselines across different settings by a substantial margin.[1]

1 Introduction

Relation extraction (RE) aims to extract semantic relations between two entities from unstructured text and is an important task in natural language processing (NLP). Formally, given an entity pair (e_1, e_2) from a knowledge base (KB) and a sentence (instance) that mentions them, RE tries to predict if a relation r from a predefined relation set exists between e_1 and e_2. A special relation *NA* is used if none of the predefined relations holds.

Given that it is costly to construct large-scale labeled instances for RE, distant supervision (DS) has been a popular strategy to automatically construct (noisy) training data. It assumes that if two entities hold a relation in a KB, all sentences mentioning them express the same relation. Noticing that the DS assumption does not always hold and has the wrong labeling problem, many efforts including (Riedel et al., 2010; Hoffmann et al., 2011; Surdeanu et al., 2012) have adopted the multi-instance learning paradigm to tackle the challenge, and more recently, neural models with attention mechanism have been proposed to de-emphasize the noisy instances (Lin et al., 2016; Ji et al., 2017; Han et al., 2018). Such models tend to work well when there are a large number of sentences talking about the target entity pair (Lin et al., 2016).

Mark Aguirre - Chicago place_lived	**mark_aguirre** and isiah thomas , the team president , flew to **chicago** to attend the funeral of ray meyer.

Mr. Basketball USA		
Mark Aguirre	Westinghouse	**Chicago, IL**
Rasheed Wallace	Simon Gratz	Philadelphia, PA
LeBron James	St. Vincent – St. Mary	Akron, OH
...	...	...

LeBron James - Akron	... including the **akron** native **lebron_james** , giving this quintessential college town ...
	... another **lebron_james** , the high school phenomenon from **akron** , ohio , who ...
Rasheed Wallace - Philly	... scored 13 points and **philadelphia** native , **rasheed_wallace** , added 10 ...
...	...

Figure 1: Illustration of *2-hop* distant supervision. The top panel shows a target entity pair, one sentence that mentions it, and the relation under study which cannot be inferred from the sentence. The middle gives part of a table from Wikipedia page "*Mr. Basketball USA*", where we can extract *anchors* for the target entity pair. The bottom shows some sentences that are associated with the anchors, which more clearly indicate the underinvestigated relation and can be utilized to extract relations between the target entity pair.

[1] Our source code and datasets are at https://github.com/sunlab-osu/REDS2.

However, we observe that there can be a large portion of entity pairs that have very few supporting sentences (e.g., nearly 75% of entity pairs in the Riedel et al. (2010) dataset only have one single sentence mentioning them), which makes distantly supervised RE even more challenging.

The conventional distant supervision strategy only exploits instances that directly mention a target entity pair, and because of this, we refer to it as *1-hop* distant supervision. On the other hand, there are a large number of Web tables that contain relational facts about entities (Cafarella et al., 2008; Venetis et al., 2011; Wang et al., 2012). Owing to the semi-structured nature of tables, we can extract from them sets of entity pairs that share common relations, and sentences mentioning these entity pairs often have similar semantic meanings. Under this observation, we introduce a new strategy named *2-hop* distant supervision: We define entity pairs that potentially have the same relation with a given target entity pair as *anchors*, which can be found through Web tables, and aim to fully exploit the sentences that mention those anchor entity pairs to augment RE for the target entity pair. Figure 1 illustrates the 2-hop DS strategy.

The intuition behind 2-hop DS is if the target entity pair holds a certain relation, one of its anchors is likely to have that relation too and at least one sentence mentioning the anchors should express the relation. Despite being noisy, the 2-hop DS can provide extra, informative supporting sentences for the target entity pair. One straightforward approach is to merge the two bags of sentences respectively derived from 1-hop and 2-hop DS as one single set and apply existing multi-instance learning models. However, the 2-hop DS strategy also has the wrong labeling problem that already exists in 1-hop DS. Simply mixing the two sets of sentences together may mislead the prediction, especially when there is a great disparity in their size. In this paper, we propose REDS2[2], a new neural relation extraction method in the multi-instance learning paradigm, and design a hierarchical model structure to fuse information from 1-hop and 2-hop DS. We evaluate REDS2 on a widely used benchmark dataset and show that it consistently outperforms various baseline models by a large margin.

We summarize our contributions as three-fold:

- We introduce 2-hop distant supervision as an extension to the conventional distant supervision, and leverage entity pairs in Web tables as anchors to find additional supporting sentences to further improve RE.
- We propose REDS2, a new neural relation extraction method based on 2-hop DS and has achieved new state-of-the-art performance in the benchmark dataset (Riedel et al., 2010).
- We release both our source code and an augmented benchmark dataset that has entity pairs aligned with those in Web tables, to facilitate future work.

[2]stands for relation extraction with 2-hop DS.

2 Related Work

Distant Supervision. One main drawback of traditional supervised relation extraction models (Zelenko et al., 2003; Mooney and Bunescu, 2006) is they require adequate amounts of annotated training data, which is time consuming and labor intensive. To address this issue, Mintz et al. (2009) proposes distant supervision (DS) to automatically label data by aligning plain text with Freebase. However, DS inevitably accompanies with the wrong labeling problem. To alleviate the noise brought by DS, Riedel et al. (2010) and Hoffmann et al. (2011) introduce multi-instance learning mechanism, which is originally used to combat the problem of ambiguously-labeled training data when predicting the activity of different drugs (Dietterich et al., 1997).

Neural Relation Extraction. Early stage relation extraction (RE) methods use features extracted by NLP tools and strongly rely on the quality of features. Due to the recent success of neural models in different NLP tasks, many researchers have investigated the possibility of using neural networks to build end-to-end relation extraction models. Zeng et al. (2014) uses convolutional neural network (CNN) to encode sentences, which is further improved through piecewise-pooling (Zeng et al., 2015). Adel and Schütze (2017) and Gupta et al. (2016) use neural networks for joint entity and relation extraction. More advanced network architectures like Tree-LSTM (Miwa and Bansal, 2016) and Graph Convolution Network (Vashishth et al., 2018) are also adopted to learn better representations by using syntactic features like dependency trees. Most recent models also incorporate neural attention technology (Lin et al., 2016) as an

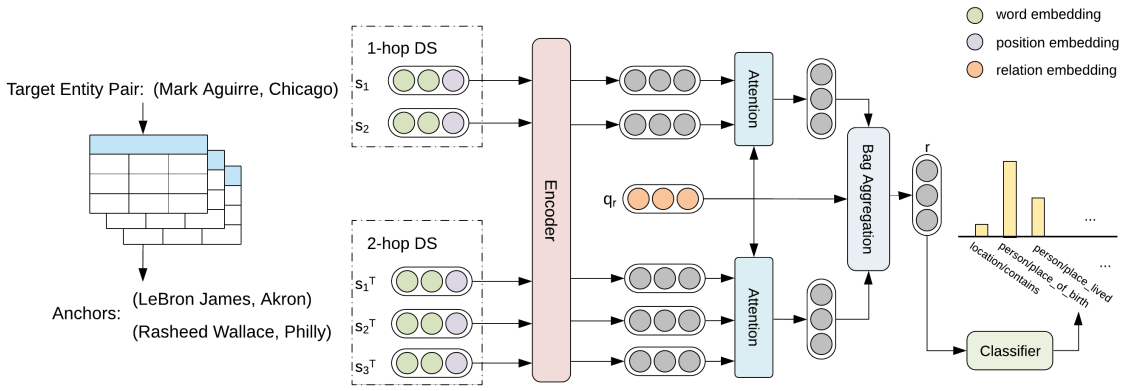


Figure 2: Overview of our method REDS2. REDS2 first obtains *anchors* of the target entity pair and constructs a 2-hop DS bag. Sentences in the 1-hop and 2-hop DS bag are individually encoded with a PCNN sentence encoder (Zeng et al., 2015). We then use selective attention and bag aggregation to get the final representation, based on which a classifier predicts scores for each candidate relation.

improvement to at-least-one multi-instance learning (Zeng et al., 2015). Han et al. (2018) further develops a hierarchical attention scheme to utilize the relation correlations and help predictions for long-tail relations.

Web Table Understanding. Aside from plain texts, there are large amounts of factual knowledge in the Web expressed in hundreds of millions of tables and other structured lists (Cafarella et al., 2008; Venetis et al., 2011), which have not been fully explored yet. Table understanding tries to match tables to KB and parse the schemas. Existing methods for table understanding mainly fall into two categories. One is based on local evidence (Venetis et al., 2011; Muñoz et al., 2014; Ritze et al., 2015). Given one table, the main idea is to first link cells to entities in KB. We can then use existing relations between linked entities to infer relations between columns and extract new facts by generalizing to all rows. However, this method requires a high overlap between table and KB, which is hampered by KB incompleteness. The other approach tries to leverage features extracted from the table header and column names (Ritze and Bizer, 2017; Cannaviccio et al., 2018). Unfortunately, a large portion of Web tables miss such metadata or contain limited information, and the second approach will fail in such cases. Although the focus of this paper is the RE task, we believe the idea of connecting Web tables and plain texts using DS can potentially benefit table understanding as well.

3 Methodology

Given a set of sentences $\mathrm{S} = \{s_1, s_2, ...\}$ and a target entity pair (h, t), we will leverage the directly associated sentence bag $S_{h,t} \subseteq \mathrm{S}$ by 1-hop distant supervision (1-hop DS bag), and the table expanded sentence bag $S^T_{h,t} \subseteq \mathrm{S}$ by 2-hop distant supervision (2-hop DS bag), for relation extraction. $S_{h,t}$ contains all instances mentioning both h and t, while $S^T_{h,t}$ is obtained indirectly through the anchors of (h, t) found in Web tables. Following previous work (Riedel et al., 2010; Hoffmann et al., 2011), we adopt the multi-instance learning paradigm to measure the probability of (h, t) having relation r.

Figure 2 gives an overview of our framework with three major components:

- **Table-aided Instance Expansion:** Given a target entity pair (h, t), we find its *anchor* entity pairs $\{(h_1, t_1), (h_2, t_2), ...\}$ through Web tables. We define an anchor entity pair as two entities co-occurring with (h, t) in some table columns at least once. $S^T_{h,t} = S_{h_1,t_1} \cup S_{h_2,t_2} \cup ...$ is then exploited to augment the directly associated bag $S_{h,t}$.

- **Sentence Encoding:** For each sentence s in bag $S_{h,t}$ or $S^T_{h,t}$, a sentence encoder is used to obtain its semantic representation $\mathbf{s}$.

- **Hierarchical Bag Aggregation:** Once the embedding of each sentence is learned, we first use a sentence-level attention mechanism to get bag representation $\mathbf{h}$ and $\mathbf{h}^T$, and

then aggregate them for final relation prediction.

3.1 Table-aided Instance Expansion

Now we introduce how to construct the table expanded sentence bag $S^T_{h,t}$ for a given target entity pair (h, t) by 2-hop distant supervision.

3.1.1 Web Tables

Web tables have been found to contain rich facts of entities and relations. It is estimated that out of a total of 14.1 billion tables on the Web, 154 million tables contain relational data (Cafarella et al., 2008) and Wikipedia alone is the source of nearly 1.6 million relational tables (Bhagavatula et al., 2015). Columns of a Wikipedia table can be classified into one of the following data types: 'empty', 'named entity', 'number', 'date expression', 'long text' and 'other' (Zhang, 2017). Here we only focus on named entity columns (NE-columns) and the Wikipedia page title, which can be easily linked to KB entities. These entities can be further categorized as:

A topic entity e^t that the table is centered around. We refer to the Wikipedia article where the table is found and take the entity it describes as e^t.

Subject entities $E^s = \{e^s_1, e^s_2, ...\}$ that can act as primary keys of the table. Following previous work on Web table analysis (Venetis et al., 2011), we select the leftmost NE-column as subject column and its entities as E^s.

Body entities $E = \{e_{1,1}, e_{1,2}, ...\}$ that compose the rest of the table. All entities in non-subject NE-columns are considered as E.

3.1.2 2-hop Distant Supervision

In the conventional distant supervision setting, each entity pair (h, t) is associated with a bag of sentences $S_{h,t}$ that directly mention h and t. The intuition behind 2-hop distant supervision is, if (h_i, t_i) and (h_j, t_j) potentially hold the same relation, we can treat them as **anchor entity pairs** for each other, and then use the 1-hop DS bag S_{h_j,t_j} to help with the prediction for (h_i, t_i) and vice versa. In this paper, we extract anchor entity pairs with the help of Web tables.

We notice that owing to the semi-structured nature of tables, (1) subject entities can usually be connected with the topic entity by the same relation. (2) Non-subject columns of a table usually have binary relationships to or are properties of the subject column. Body entities in the same column share common relations with their corresponding subject entities. For example, in Figure 1, the topic entity is "*Mr. Basketball USA*"; column 1 is the subject column and contains a list of winners of "*Mr. Basketball USA*"; column 2 and column 3 are high school and city of the subject entity.

Formally, we consider two entity pairs (h_i, t_i) and (h_j, t_j) as anchored if there exists a Web table such that either criterion below is met:

- $h_i = h_j = e^t$ and t_i, $t_j \in E^s$.
- $h_i \in E^s$ or $t_i \in E^s$, (h_i, h_j) is in the same column (and so is (t_i, t_j)), and (h_i, t_i) is in the same row (and so is (h_j, t_j))

The 2-hop DS bag $S^T_{h,t}$ is then constructed as the union of S_{h_i,t_i}'s, where (h_i, t_i) is an anchor entity pair of (h, t).

3.2 Sentence Encoding

Given a sentence s consisting of n words $s = \{w_1, w_2, ..., w_n\}$, we use a neural network with an embedding layer and an encoding layer to obtain its low-dimensional vector representation.

3.2.1 Embedding Layer

Each token is first fed into an embedding layer to embed both semantic and positional information.

Word Embedding maps words to vectors of real numbers which preserve syntactic and semantic information of words. Here we get a vector representation $\mathbf{w}_i \in \mathbb{R}^{k_w}$ for each word from a pre-trained word embedding matrix.

Position Embedding was proposed by Zeng et al. (2014). Position embedding is used to embed the positional information of each word relative to the head and tail mention. A position embedding matrix is learned in training to compute position representation $\mathbf{p}_i \in \mathbb{R}^{k_p \times 2}$.

Finally, we concatenate the word representation $\mathbf{w}_i$ and position representation $\mathbf{p}_i$ to build the input representation $\mathbf{x}_i \in \mathbb{R}^{k_i}$ (where $k_i = k_w + k_p \times 2$) for each word w_i.

3.2.2 Encoding Layer

A sequence of input representations $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, ...\}$ with a variable length is then fed through the encoding layer and converted to a fixed-sized sentence representation $\mathbf{s} \in \mathbb{R}^{k_h}$. There are many existing neural architectures that can serve as the encoding layer, such as CNN

(Zeng et al., 2014), PCNN (Zeng et al., 2015) and LSTM-RNN (Miwa and Bansal, 2016). We simply adopt PCNN here, which has been shown very powerful and efficient by a number of previous RE works.

PCNN is an extension to CNN, which first slides a convolution kernel with a window size m over the input sequence to get the hidden vectors:

$$\mathbf{h}_i = \mathrm{CNN}(\mathbf{x}_{i-\frac{m-1}{2}:i+\frac{m-1}{2}}), \tag{1}$$

A piecewise max-pooling is then applied over the hidden vectors:

$$\begin{aligned}
[\mathbf{s}^{(1)}]_j &= \max_{1\le i\le i_1}\{[\mathbf{h}_i]_j\},\\
[\mathbf{s}^{(2)}]_j &= \max_{i_i+1\le i\le i_2}\{[\mathbf{h}_i]_j\},\\
[\mathbf{s}^{(3)}]_j &= \max_{i_2+1\le i\le n}\{[\mathbf{h}_i]_j\},
\end{aligned} \tag{2}$$

where i_1 and i_2 are head and tail positions. The final sentence representation $\mathbf{s}$ is composed by concatenating these three pooling results $\mathbf{s} = [\mathbf{s}^{(1)}; \mathbf{s}^{(2)}; \mathbf{s}^{(3)}]$.

3.3 Hierarchical Bag Aggregation

After we get sentence representations $\{\mathbf{s}_1, \mathbf{s}_2, ...\}$ and $\{\mathbf{s}_1^T, \mathbf{s}_2^T, ...\}$ for S and S^T, to fuse key information from these two bags, we adopt a hierarchical aggregation design to obtain the final representation $\mathbf{r}$ for prediction. We first get bag representation $\mathbf{h}$ and $\mathbf{h}^T$ using a sentence-level selective attention, and then employ a bag-level aggregation to compute $\mathbf{r}$.

3.3.1 Sentence-level Selective Attention

Since the wrong labeling problem inevitably exists in both 1-hop and 2-hop distant supervision, here we use selective attention to assign different weights to different sentences given relation r and de-emphasize the noisy sentences. The attention is caculated as follows:

$$\begin{aligned}
e_i &= \mathbf{q}_r^{\mathsf{T}}\mathbf{s}_i,\\
\alpha_i &= \frac{\exp(e_i)}{\sum_{j=1}^{n}\exp(e_j)},\\
\mathbf{h} &= \sum_{i=1}^{n}\alpha_i\mathbf{s}_i,
\end{aligned} \tag{3}$$

where $\mathbf{q}_r$ is a query vector assigned to relation r. $\mathbf{h}$ and $\mathbf{h}^T$ are computed respectively for the two bags S and S^T.

3.3.2 Bag-level Aggregation

Since 2-hop DS bag S^T is collected indirectly through anchor entity pairs in Web tables, despite that it brings abundant information, it also contains a massive amount of noise. Thus treating S^T equally as S may mislead the prediction, especially when their sizes are extremely imbalanced.

To automatically decide how to balance between S and S^T, we utilize information from $\mathbf{h}$, $\mathbf{h}^T$ and $\mathbf{q}_r$ to predict a weight β:

$$\beta = \sigma(\mathbf{W}[\mathbf{h}; \mathbf{h}^T; \mathbf{q}_r] + \mathbf{b}), \tag{4}$$

where vector $\mathbf{W}$ and scalar $\mathbf{b}$ are learnable variables and σ is the sigmoid function. Next, β is used as a weight to fuse information from 1-hop DS and 2-hop DS, determined by S and S^T of the current target entity pair and relation r. We then obtain the final representation $\mathbf{r}$ as:

$$\mathbf{r} = \beta\mathbf{h} + (1-\beta)\mathbf{h}^T, \tag{5}$$

Finally, we define the conditional probability $P(r|S, S^T, \theta)$ as follows,

$$P(r|S, S^T, \theta) = \frac{\exp(\mathbf{o}_r)}{\sum_{k=1}^{n_r}\exp(\mathbf{o}_k)} \tag{6}$$

where $\mathbf{o}$ is the score vector for current target entity pair having each relation,

$$\mathbf{o} = \mathbf{M}\mathbf{r} + \mathbf{d}, \tag{7}$$

here $\mathbf{M}$ is the representation matrix of relations, which shares weights with $\mathbf{q}_r$'s. $\mathbf{d}$ is a learnable bias term.

3.4 Optimization

We adopt the cross-entropy loss as the training objevtive function. Given a set of target entity pairs with relations $\pi = \{(h_1, t_1, r_1), (h_2, t_2, r_2), ...\}$, we define the loss function as follows:

$$J(\theta) = -\frac{1}{|\pi|}\sum_{i=1}^{|\pi|}\log P(r_i|S_{h_i,t_i}, S^T_{h_i,t_i}, \theta). \tag{8}$$

All models are trained with stochastic gradient descent (SGD) to minimize the objective function. The same sentence encoder is used to encode S and S^T.

	Train		Test	
	Overall	Non-NA	Overall	Non-NA
# Entity Pairs	291699	18144	96678	1761
# Entity Pairs with $\|S^T\| > 0$	17565	6928	4832	824
% Entity Pairs with $\|S^T\| > 0$	6.02	**38.18**	5.00	**46.79**
mean $\|S\|$	1.69	5.24	1.62	2.78
mean $\|S^T\|$	147.65	190.61	131.65	217.23

Table 1: Dataset statistics. We show statistics of entity pairs that hold non-NA relations separately from overall, as they are important relational facts to discover. Among non-NA entity pairs, 38.18% in training and 46.79% in testing have nonempty S^T, which respectively have 190.61 and 217.23 sentences on average.

4 Experiments

4.1 Datasets and Evaluation

We evaluate our model on the New York Times (NYT) dataset developed by Riedel et al. (2010), which is widely used in recent works. The dataset has 53 relations including a special relation NA which indicates none of the other 52 relations exists between the head and tail entity.

We use the WikiTable corpus collected by Bhagavatula et al. (2015) as our table source. It originally contains around 1.65M tables extracted from Wikipedia pages. Since the NYT dataset is already linked to Freebase, we perform entity linking on the table cells and the Wikipedia page titles using existing mapping from Wikipedia URL to Freebase MID (Machine Identifier). We then align the table corpus with NYT and construct S^T for entity pairs as detailed in section 3.1. *For both training and testing, we only use entity pairs and sentences in the original NYT training data for table-aided instance expansion.* We set the max size of S^T as 300, and randomly sample 300 sentences if $|S^T| > 300$. Statistics of our final dataset is summarized in Table 1. One can see that 38.18% and 46.79% of relational facts (i.e., entity pairs holding non-NA relations) respectively in the training and testing set can potentially benefit from leveraging 2-hop DS.

Following prior work (Mintz et al., 2009), we use the testing set for held-out evaluation, and evaluate models by comparing the predicted relational facts with those in Freebase. For evaluation, we rank the extracted relational facts based on model confidence and plot precision-recall curves. In addition, we also show the area under the curve (AUC) and precision values at specific recall rates to conduct a more comprehensive comparison.

Window size m	3
Sentence Representation Size k^h	230
Word Dimension k^w	50
Position Dimension k^p	5
Pre-train Learning Rate λ_P	0.005
Fine-tune Learning Rate λ_F	0.002
Dropout Probability p	0.5

Table 2: Parameter settings in REDS2.

4.2 Baselines

We compare REDS2 with the following baselines:
PCNN+ATT (Lin et al., 2016). This model uses a PCNN encoder combined with selective attention over sentences. Since this is also the base block of our model, we also refer to it as BASE in this paper.
PCNN+HATT (Han et al., 2018). This is another PCNN based relation extraction model, where the authors use hierarchical attention to model the semantic correlations among relations.
RESIDE (Vashishth et al., 2018). It uses Graph Convolutional Networks (GCN) for sentence encoding, and also leverages relevant side information like relation alias and entity type.

Results of PCNN+HATT and RESIDE are directly taken from the code repositories released by the authors. For PCNN+ATT, we report results obtained by our reproduced model, which are close to those shown in (Lin et al., 2016). To simply verify the effectiveness of adding extra supporting sentences from 2-hop DS, we also compare the following vanilla method with PCNN+ATT:
BASE+MERGE. For each target entity pair (h, t), we simply merge S and S^T as one sentence bag, and apply the trained PCNN+ATT (or, BASE) model.

4.3 Implementation Details

We preprocess the WikiTable corpus with PySpark to build index for anchor entity pairs. On a single machine with two 8-core E5 CPUs and 256 GB memory, this processing takes around 20 minutes.

We use word embeddings from (Lin et al., 2016) for initialization, which are learned by word2vec tool[3] on NYT corpus. The vocabulary is composed of words that appear more than 100 times in the corpus and words in an entity mention are concatenated as a single word.

[3] https://code.google.com/archive/p/word2vec/

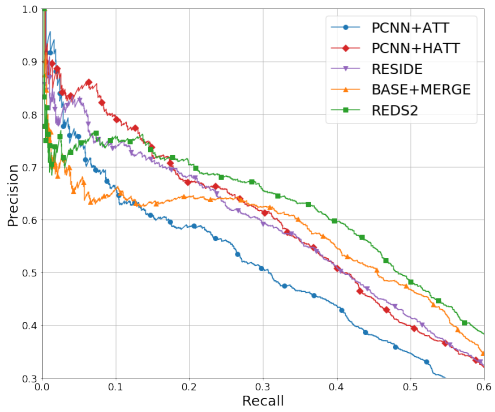


Figure 3: Precision-recall curves for the proposed model and various baselines.

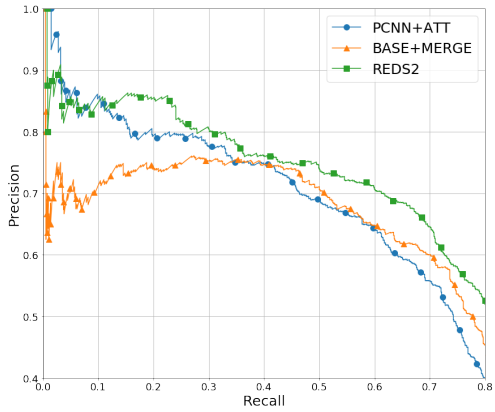


Figure 4: Precision-recall curves on the subset of test entity pairs whose S^T is not empty, to better show the effect of hierarchical bag aggregation design.

Method	P@0.1	P@0.2	P@0.3	AUC
PCNN+ATT	65.9	58.7	50.4	35.8
PCNN+HATT	**79.3**	67.2	61.6	42.0
RESIDE	73.6	68.4	59.5	41.5
BASE+MERGE	65.9	64.5	62.4	41.2
REDS2	75.9	**70.4**	**65.5**	**44.7**

Table 3: Comparison on Precision@recall and AUC.

To see the effect of 2-hop DS more directly, we set most parameters in REDS2 following Lin et al. (2016). Since the original NYT dataset only contains training and testing set, we randomly sample 20% training data for development. We first pre-train a PCNN+ATT model with only S and sentence-level selective attention. This BASE model converges in around 100 epochs. We then fine-tune the entire model with S^T and bag-level aggregation added, which can finish within 50 epochs. Some key parameter settings in REDS2 are summarized in Table 2.

In testing phase, inference using 2-hop DS is slower, because the average size of S^T is about 100 times that of S. With single 2080ti GPU, one full pass of testing data takes around 37s using REDS2, compared with 12s using BASE model.

4.4 Results

4.4.1 Overall Evaluation Results

Evaluation results on all target entity pairs in testing set are shown in Figure 3 and Table 3, from which we make the following observations:

(1) Figure 3 shows all models obtain a reasonable precision when recall is smaller than 0.05. With the recall gradually increasing, the performance of models with 2-hop DS drops slower than those existing methods without. From Figure 3, we can see simply merging S^T with S in BASE+MERGE can boost the performance of basic PCNN+ATT model, and even achieves higher precision than state-of-the-art models like PCNN+HATT when recall is greater than 0.3. This demonstrates that models utilizing 2-hop DS are more robust and remain a reasonable precision when including more lower-ranked relational facts which tend to be more challenging to predict because of insufficient evidence.

(2) As shown in both Figure 3 and Table 3, REDS2 achieves the best results among all the models. Even when compared with PCNN+HATT and RESIDE which adopt extra relation hierarchy and side information from KB, our model still enjoys a significant performance gain. This is because our method can take advantage of the rich entity pair correlations in Web tables and leverage the extra information brought by 2-hop DS. We anticipate our REDS2 model can be further improved by using more advanced sentence encoders and extra mechanisms like reinforcement learning (Feng et al., 2018) and adversarial training (Wu et al., 2017), which we leave for future work.

4.4.2 Effect of Hierarchical Bag Aggregation

To further show the effect of our hierarchical bag aggregation design, here we also plot precision-recall curves in Figure 4 on a subset of entity pairs in the test set (i.e., 4832 in total according to Table 1) whose table expanded sentence bag S^T is not empty.

One main challenge of using 2-hop DS is it brings more noise. As shown in Table 1, for en-

Test Mode	SINGLE				MULTIPLE ONE				MULTIPLE TWO				MULTIPLE ALL			
Metric	P@0.1	P@0.2	P@0.3	AUC	P@0.1	P@0.2	P@0.3	AUC	P@0.1	P@0.2	P@0.3	AUC	P@0.1	P@0.2	P@0.3	AUC
PCNN+ATT	57.3	53.0	41.3	30.0	72.9	66.4	57.5	39.0	75.3	69.7	63.8	45.9	80.5	68.6	64.2	48.6
RESIDE	66.9	60.5	51.8	36.2	79.7	70.4	57.4	42.1	77.7	76.1	67.6	46.5	83.2	**80.3**	73.2	51.9
PCNN+HATT	**70.1**	59.1	50.8	34.8	74.5	67.4	57.2	40.1	78.4	68.9	65.9	44.5	**86.4**	75.5	70.0	49.7
BASE+MERGE	62.2	58.5	55.2	35.6	71.4	74.9	73.2	51.3	70.0	71.8	73.9	52.0	70.7	71.1	73.2	52.0
REDS2	69.1	**61.1**	**57.4**	**37.5**	**82.4**	**81.9**	**78.1**	**56.3**	**81.4**	**79.6**	**76.9**	**57.2**	82.4	79.6	**76.6**	**57.6**

Table 4: Comparison on Precision@recall and AUC under different testing settings, detailed in Section 4.4.3.

max $\lvert S^T \rvert$	P@0.1	P@0.2	P@0.3	AUC
10	57.9	55.8	51.5	36.2
50	69.4	65.7	60.8	42.2
100	70.4	66.7	62.3	43.2
200	72.8	68.4	63.4	44.0
300	**75.9**	**70.4**	**65.5**	**44.7**

Table 5: Effect of the table expanded sentence bag size $|S^T|$ on Precision@recall and AUC.

Relation: `country.capital`	
1-hop	... the golden gate bridge and the petronas towers in **kuala_lumpur**, **malaysia**, was experienced ...
2-hop	a friend from **cardiff** , the capital city of **wales** , lives for complex ...

Table 6: An example for case study, where the sentence with the highest attention weight is selected respectively from 1-hop and 2-hop sentence bag.

tity pair with nonempty S^T, the size of S^T is usually tens of times the size of S. From Figure 4 we can see BASE+MERGE performs much worse compared with PCNN+ATT when recall is smaller than 0.2. This is because 2-hop DS bag tends to be much larger than 1-hop DS bag, and the model has a larger chance to attend to the noisy sentences obtained from 2-hop DS. While ignoring the information in its directly associated sentences. We alleviate this problem by introducing hierarchical structure to first aggregate the two sets separately and then weight and sum them together. The proposed `REDS2` model has a comparable precision with PCNN+ATT in the beginning and gradually outperform it.

4.4.3 Effect of Sentence Number

Number of sentences from 1-hop DS. In the originally testing set, there are 79176 entity pairs that are associated with only one sentence, out of which 1149 actually have relations. We hope our model can improve performance on these long-tail entities. Following Lin et al. (2016), we design the following test settings to evaluate the effect of sentence number: the "SINGLE" test setting contains all entity pairs that correspond to only one sentence; the "MULTIPLE" test setting contains the rest of entity pairs that have at least two sentences associated. We further construct the "ONE" testing setting where we randomly select one sentence for each entity pair; the "TWO" setting where we randomly select two sentences for each entity pair and the "ALL" setting where we use all the associated sentences from MULTIPLE. We use all sentences in S^T for each entity pair if it is nonempty. Results are shown in Table 4, from which we can see that `REDS2` and BASE+MERGE have 25.0% and 18.7% improvements under AUC compared with PCNN+ATT in the SINGLE setting. Although the performance of all models generally improves as the sentence number increases in MULTIPLE setting, models leveraging 2-hop DS are more stable and have smaller changes. These observations indicate that 2-hop DS is helpful when information obtained by 1-hop DS is insufficient.

Number of sentences from 2-hop DS. We also evaluate how the number of sentences obtained by 2-hop DS will affect the performance of our proposed model. In Table 5, we show the performance of `REDS2` with different numbers of sentences sampled from S^T. We observe that: (1) Performance of `REDS2` improves as the number of sentences sampled increases. This shows that the selective attention over S^T can effectively take advantage of the extra information from 2-hop DS while filtering out noisy sentences. (2) Even with 50 randomly sampled sentences, our model `REDS2` still has a higher AUC than all baselines in Table 3. This indicates information obtained by 2-hop DS is redundant, even a small portion can be beneficial to relation extraction. How to sample a representative set effectively is worth further exploring in future work.

4.5 RE for Entity Pairs with Empty 1-hop Sentence Bag

We observe that there are large amounts of entity pairs in the table corpus that have no associated sentences but have anchor entity pairs mentioned in the text corpus. By leveraging 2-hop distant supervision, we can do relation extraction for this set of entity pairs.

We extract a total number of 251917 entity pairs from the WikiTable dataset which do not exist in the NYT dataset but have at least one anchor entity pair that appear in the original NYT training data. We randomly sample 10000 examples and evaluate our trained model on them. Surprisingly, the relation extraction result is even better than the result on the NYT test data in Table 3, with an overall AUC of 54.7 and a P@0.3 of 71.1. This can be explained partly by two observations: (1) The table corpus generates higher-quality entity pairs, 18% of extracted entity pairs have non-NA relations, compared with only 1.8% in NYT test data. (2) The newly extracted entity pairs have 14 useful anchor entity pairs and 175 2-hop DS sentences on average, which give ample information for prediction. This study shows that for two entities that have no directly associated sentences, it is possible to utilize the 2-hop DS to predict their relations accurately.

4.6 Case Study and Error Analysis

In addition to the motivating example from the training set shown in Figure 1, we also demonstrate how 2-hop DS helped relation extraction using an example from the testing set in Table 6. As we can see, the sentence with the highest attention weight in 1-hop DS bag does not express the desired relation between the target entity pair whereas that in 2-hop DS bag clearly indicates the `country.capital` relation.

We also conduct an error analysis by analyzing examples where REDS2 gives worse predictions than BASE (e.g., assigns a lower score to a correct relation or a higher score to a wrong relation), and 50 examples with most disparity in the two methods' scores are selected. We find that 29 examples have wrong labels caused by KB incompleteness and our model in fact makes the right prediction. 11 examples are due to errors in column processing (e.g., errors in NE/subject column selection and entity linking), 9 are caused by anchor entity pairs with differet relations (e.g., **(Greece, Atlanta)** and **(Mexico, Xalapa)** are in the same table *"National Records in High Jump"* under columns **(Nation, Place)**, but only the latter has relation `location.contains`), and 1 is because of wrong information in the original table.

5 Conclusion and Future Work

This paper introduces 2-hop distant supervision for relation extraction, based on the intuition that entity pairs in relational Web tables often share common relations. Given a target entity pair, we define and find its anchor entity pairs via Web tables and collect all sentences that mention the anchor entity pairs to help relation prediction. We develop a new neural RE method `REDS2` in the multi-instance learning paradigm which fuses information from 1-hop DS and 2-hop DS using a hierarchical model structure, and substantially outperforms existing RE methods on a benchmark dataset. Interesting future work includes: (1) Given that information from 2-hop DS is redundant and noisy, we can explore smarter sampling and/or better bag-level aggregation methods to capture the most representative information. (2) Metadata in Web tables like headers and column names also contain rich information, which can be incorporated to further improve RE performance.

6 Acknowledgments

This research was sponsored in part by the Army Research Office under cooperative agreements NSF Grant IIS1815674, W911NF-17-1-0412, Fujitsu gift grant, and Ohio Supercomputer Center [8]. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Heike Adel and Hinrich Schütze. 2017. Global normalization of convolutional neural networks for joint entity and relation classification. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1723–1729.

Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. Tabel: entity linking in web

tables. In *International Semantic Web Conference*, pages 425–441.

Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549.

Matteo Cannaviccio, Lorenzo Ariemma, Denilson Barbosa, and Paolo Merialdo. 2018. Leveraging wikipedia table schemas for knowledge graph augmentation. In *Proceedings of the 21st International Workshop on the Web and Databases*, page 5. ACM.

Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. 1997. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2):31–71.

Jun Feng, Minlie Huang, Li Zhao, Yang Yang, and Xiaoyan Zhu. 2018. Reinforcement learning for relation classification from noisy data. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Pankaj Gupta, Hinrich Schütze, and Bernt Andrassy. 2016. Table filling multi-task recurrent neural network for joint entity and relation extraction. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2537–2547.

Xu Han, Pengfei Yu, Zhiyuan Liu, Maosong Sun, and Peng Li. 2018. Hierarchical relation extraction with coarse-to-fine grained attention. In *Proceedings of EMNLP*.

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 541–550. Association for Computational Linguistics.

Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. 2017. Distant supervision for relation extraction with sentence-level attention and entity descriptions. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Yankai Lin, Shiqi Shen, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. 2016. Neural relation extraction with selective attention over instances. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2124–2133.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using lstms on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1105–1116.

Raymond J Mooney and Razvan C Bunescu. 2006. Subsequence kernels for relation extraction. In *Advances in neural information processing systems*, pages 171–178.

Emir Muñoz, Aidan Hogan, and Alessandra Mileo. 2014. Using linked data to mine rdf from wikipedia's tables. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 533–542. ACM.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling relations and their mentions without labeled text. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 148–163. Springer.

Dominique Ritze and Christian Bizer. 2017. Matching web tables to dbpedia - A feature utility study. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.*, pages 210–221.

Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching html tables to dbpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, page 10. ACM.

Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D Manning. 2012. Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 455–465.

Shikhar Vashishth, Rishabh Joshi, Sai Suman Prayaga, Chiranjib Bhattacharyya, and Partha Talukdar. 2018. Reside: Improving distantly-supervised neural relation extraction using side information. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1257–1266.

Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment*, 4(9):528–538.

Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q Zhu. 2012. Understanding tables on the web. In *International Conference on Conceptual Modeling*, pages 141–155. Springer.

Yi Wu, David Bamman, and Stuart Russell. 2017. Adversarial training for relation extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1778–1783.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. 2003. Kernel methods for relation extraction. *Journal of machine learning research*, 3(Feb):1083–1106.

Daojian Zeng, Kang Liu, Yubo Chen, and Jun Zhao. 2015. Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1753–1762.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation classification via convolutional deep neural network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2335–2344.

Ziqi Zhang. 2017. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, 8(6):921–957.

Adversarial Training for Code Retrieval with Question-Description Relevance Regularization

Jie Zhao
The Ohio State University
zhao.1359@osu.edu

Huan Sun
The Ohio State University
sun.397@osu.edu

Abstract

Code retrieval is a key task aiming to match natural and programming languages. In this work, we propose adversarial learning for code retrieval, that is regularized by question-description relevance. First, we adapt a simple adversarial learning technique to generate difficult code snippets given the input question, which can help the learning of code retrieval that faces bi-modal and data-scarce challenges. Second, we propose to leverage question-description relevance to regularize adversarial learning, such that a generated code snippet should contribute more to the code retrieval training loss, only if its paired natural language description is predicted to be less relevant to the user given question. Experiments on large-scale code retrieval datasets of two programming languages show that our adversarial learning method is able to improve the performance of state-of-the-art models. Moreover, using an additional duplicate question prediction model to regularize adversarial learning further improves the performance, and this is more effective than using the duplicated questions in strong multi-task learning baselines.[1]

1 Introduction

Recently there has been a growing research interest in the intersection of natural language (NL) and programming language (PL), with exemplar tasks including code generation (Agashe et al., 2019; Bi et al., 2019), code summarizing (LeClair and McMillan, 2019; Panthaplackel et al., 2020), and code retrieval (Gu et al., 2018). In this paper, we study code retrieval, which aims to retrieve code snippets for a given NL question such as *"Flatten a shallow list in Python."* Advanced code retrieval tools can save programmers tremendous time in various scenarios, such as how to fix a bug, how to implement a function, which API to use, etc. Moreover, even if the retrieved code snippets do not perfectly match the NL question, editing them is often much easier than generating a code snippet from scratch. For example, the retrieve-and-edit paradigm (Hayati et al., 2018; Hashimoto et al., 2018; Guo et al., 2019) for code generation has attracted growing attention recently, which first employs a code retriever to find the most relevant code snippets for a given question, and then edit them via a code generation model. Previous work has shown that code retrieval performance can significantly affect the final generated results (Huang et al., 2018) in such scenarios.

There have been two groups of work on code retrieval: (1) One group of work (e.g., the recent retrieve-and-edit work (Hashimoto et al., 2018; Guo et al., 2019)) assumes each code snippet is associated with NL descriptions and retrieves code snippets by measuring the relevance between such descriptions and a given question. (2) The other group of work (e.g., CODENN (Iyer et al., 2016) and Deep Code Search (Gu et al., 2018)) directly measures the relevance between a question and a code snippet. Comparing with the former group, this group of work has the advantage that they can still apply when NL descriptions are not available for candidate code snippets, as is often the case for many large-scale code repositories (Dinella et al., 2020; Chen and Monperrus, 2019). Our work connects with both groups: We aim to directly match a code snippet with a given question, but during training, we will utilize question-description relevance to improve the learning process.

Despite the existing efforts, we observe two challenges for directly matching code snippets with NL questions, which motivate this work. First, code retrieval as a bi-modal task requires representation learning of two heterogeneous but com-

[1] Source code and dataset are available at https://github.com/jiez-osu/QQC.

plementary modalities, which has been known to be difficult (Cvitkovic et al., 2019; LeClair and McMillan, 2019; Akbar and Kak, 2019) and may require more training data. This makes code retrieval more challenging compared to document retrieval where the target documents often contain useful shallow NL features like keywords or key phrases. Second, code retrieval often encounters special one-to-many mapping scenarios, where one NL question can be solved by multiple code solutions that take very different approaches. Table 1 illustrates the challenges. For i=1,2 or 3, $q^{(i)}$ is an NL question/description that is associated with a Python answer $c^{(i)}$. Here, question $q^{(1)}$ should be matched with multiple code snippets: $c^{(1)}$ and $c^{(2)}$, because they both flatten a 2D list despite with different programming approaches. In contrast, $c^{(3)}$ is performing a totally different task, but uses many overlapped tokens with $c^{(1)}$. Hence, it can be difficult to train a code retrieval model that generalizes well to match $q^{(1)}$ with both $c^{(1)}$ and $c^{(2)}$, and is simultaneously able to distinguish $c^{(1)}$ from $c^{(3)}$.

To address the first challenge, we propose to introduce adversarial training to code retrieval, which has been successfully applied to transfer learning from one domain to another (Tzeng et al., 2017) or learning with scarce supervised data (Kim et al., 2019). Our intuition is that by employing a generative adversarial model to produce *challenging negative code snippets* during training, the code retrieval model will be strengthened to distinguish between positive and negative $\langle q, c\rangle$ pairs. In particular, we adapt a generative adversarial sampling technique (Wang et al., 2017), whose effectiveness has been shown in a wide range of uni-modal text retrieval tasks.

For the second challenge, we propose to further employ *question-description (QD) relevance* as a complementary uni-modal view to reweight the adversarial training samples. In general, our intuition is that the code retrieval model should put more weights on the adversarial examples that are hard to distinguish by itself, but easy from the view of a QD relevance model. This design will help solve the one-to-many issue in the second challenge, by differentiating true negative and false negative adversarial examples: If a QD relevance model also suggests that a code snippet is not relevant to the original question, it is more likely to be a true negative, and hence the code retrieval model should put more weights on it. Note that this QD relevance design aims to help train the code retrieval model better and we do not need NL descriptions to be associated with code snippets at testing phase.

$q^{(1)}$	*Flatten a shallow list in Python*
$c^{(1)}$	`from itertools import chain` `rslt = chain(*list_2d)`
$q^{(2)}$	*How to flatten a 2D list to 1D without using numpy?*
$c^{(2)}$	`list_of_lists = [[1,2,3],[1,2],[1,4,5,6,7]]` `[j for sub in list_of_lists for j in sub]`
$q^{(3)}$	*How to get all possible combinations of a list's elements?*

$c^{(3)}$	`from itertools import chain, combinations`
	`subsets = chain(*map(lambda x: combinations(`
	`mylist, x), range(0, len(mylist)+1)))`

Table 1: Motivating Example. $\langle q^{(i)}, c^{(i)}\rangle$ denotes an associated ⟨natural language question, code snippet⟩ pair. $q^{(i)}$ can also be viewed as a description of $c^{(i)}$. Given $q^{(1)}$, the ideal code retrieval result is to return both $c^{(1)}$ and $c^{(2)}$ as their programming semantics are equivalent. Contrarily, $c^{(3)}$ is semantically irrelevant to $q^{(1)}$ and should not be returned, although its surface form is similar to $c^{(1)}$. In such cases, it can be easier to decide their relationships from the question perspective, because $\langle q^{(1)}, q^{(2)}\rangle$ are more alike than $\langle q^{(1)}, q^{(3)}\rangle$.

We conduct extensive experiments using a large-scale ⟨question, code snippet⟩ dataset StaQC (Yao et al., 2018) and our collected duplicated question dataset from Stack Overflow[2]. The results show that our proposed learning framework is able to improve the state-of-the-art code retrieval models and outperforms using adversarial learning without QD relevance regularization, as well as strong multi-task learning baselines that also utilize question duplication data.

2 Overview

The work studies ***code retrieval***, a task of matching questions with code, which we will use **QC** to stand for. The training set $\mathcal{D}^{\text{QC}}$ consists of NL question and code snippet pairs $\mathcal{D}^{\text{QC}}$=$\{q^{(i)}, c^{(i)}\}$. Given NL question $q^{(i)}$, the QC task is to find $c^{(i)}$ from $\mathcal{D}^{\text{QC}}$ among all the code snippets. For simplicity, we omit the data sample index and use q and c to denote a QC pair, and c^{-} to represent any other code snippets in the dataset except for c.

Our goal is to learn a QC model, denoted as f_θ^{QC}, that retrieves the highest score code snippets for an input question: $\arg\max_{c'\in\{c\}\cup\{c^-\}} f_\theta^{\text{QC}}(q, c')$. Note that at testing time, the trained QC model f^{QC} can be used to retrieve code snippets from any code bases, unlike the group of QC methods (Hayati et al., 2018; Hashimoto et al., 2018; Guo et al.,

[2] https://stackoverflow.com/

2019) relying on the availability of NL descriptions of code.

We aim to address the aforementioned challenges in code retrieval through two strategies: (1) We introduce adversarial learning (Goodfellow et al., 2014a) to alleviate the bi-modal learning challenges. Specifically an adversarial QC generator selects unpaired code snippets that are difficult for the QC model to discriminate, to strengthen its ability to distinguish top-ranked positive and negative samples (Wang et al., 2017). (2) We also propose to employ a question-description (**QD**) relevance model to provide a secondary view on the generated adversarial samples, inspired by the group of QC work that measures the relevance of code snippets through their associated NL descriptions.

Figure 1 gives an overview of our proposed learning framework, which does not assume specific model architectures and can be generalized to different base QC models or use different QD relevance models. A general description is given in the caption. In summary, the adversarial QC generator selects $\hat{c}$ that is unpaired with a given q. $\hat{q}$ is an NL description of $\hat{c}$. Details on how to acquire $\hat{q}$ will be introduced in Section 3.2. Next, a QD model predicts a relevance score for $\langle q, \hat{q} \rangle$. A pairwise ranking loss is calculated based on whether the QC model discriminates ground-truth QC pair $\langle q, c \rangle$ from unpaired $\langle q, \hat{c} \rangle$. Learning through this loss is reweighted by a down-scale factor, which is dynamically determined by the QD relevance prediction score. This works as a regularization term over potential false negative adversarial samples.

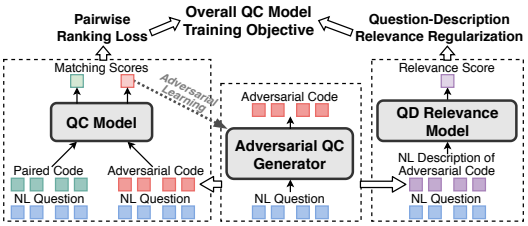


Figure 1: Regularized adversarial learning framework. Best viewed in color. The adversarial QC generator (middle) produces an adversarial code given an NL question. The QD relevance model (right) then predicts a relevance score between the given question and the NL description or the generated adversarial code. A pairwise ranking loss is computed between the ground-truth code and the adversarial code. The QC model (left) is trained with the ranking loss, after it is scaled by a QD relevance regularization weight that depends on the QD relevance score. The parameter update is larger when the relevance score is smaller and vice versa.

3 Proposed Methodology

We now introduce in detail our proposed learning framework. We start with the adversarial learning method in Section 3.1 and then discuss the rationale to incorporate question-description or QD relevance feedback in Section 3.2, before putting them together in Section 3.3 and Section 3.4.

3.1 Adversarial Learning via Sampling

We propose to apply adversarial learning (Goodfellow et al., 2014a) to code retrieval. Our goal is to train a better QC model f_θ^{QC} by letting it play the adversarial game with a QC generator model g_ϕ^{QC}. θ represents the parameters of the QC model and ϕ represents the parameters of the adversarial QC generator. As in standard adversarial learning, f_θ^{QC} plays the discriminator role to distinguish ground-truth code snippet c from generated pairs $\hat{c}$. The training objective of the QC model is to minimize $\mathcal{L}_\theta$ below:

$$\mathcal{L}_\theta = \sum_i \mathbb{E}_{\hat{c} \sim P_\phi(c|q^{(i)})} l_\theta(q^{(i)}, c^{(i)}, \hat{c}),$$

$$l_\theta = \max(0, d + f_\theta^{\text{QC}}(q^{(i)}, \hat{c}) - f_\theta^{\text{QC}}(q^{(i)}, c^{(i)})),$$

where l_θ is a pairwise ranking loss, and specifically we use a hinge loss with margin d. $\hat{c}$ is generated by g_ϕ^{QC} and follows a probability distribution $P_\phi(c|q^{(i)})$. g_ϕ^{QC} aims to assign higher probabilities to code snippets that would mislead f_θ^{QC}.

There are many ways to realize the QC generator. For example, one may employ a sequence model to generate the adversarial code snippet $\hat{c}$ token by token (Bi et al., 2019; Agashe et al., 2019). However, training a sequence generation model is difficult, because the search space of all code token combinations is huge. Henceforce, we turn to a simpler idea inspired by Wang et al. (2017), and restrict the generation of $\hat{c}$ to the space of all the existing code snippets in the training dataset $\mathcal{D}^{\text{QC}}$. The QC generator then only needs to sample an existing code snippet $c^{(j)}$ from an adversarial probability distribution conditioned on a given query and let it be $\hat{c}$, i.e., $\hat{c}{=}c^{(j)}{\sim}P_\phi(c|q^{(i)})$. Adopting this method will make training the QC generator easier, and ensures that the generated code snippets are legitimate as they directly come from the training dataset. We

define the adversarial code distribution as:

$$P_\phi(c|q^{(i)}) = \frac{\exp(g_\phi^{\text{QC}}(q^{(i)}, c)/\tau)}{\sum_{c'} \exp(g_\phi^{\text{QC}}(q^{(i)}, c')/\tau)} ,$$

where g_ϕ^{QC} represents an adversarial QC matching function. τ is a temperature hyper-parameter used to tune the distribution to concentrate more of less on top-scored code snippets. Moreover, scoring all code snippets can be computationally inefficient in practice. Therefore, we use the method of Yang et al. (2019) to first uniformly sample a subset of data, whose size is much smaller than the entire training set size, and then perform adversarial sampling on this subset.

The generator function g_ϕ^{QC} can be pre-trained in the same way as the discriminator (i.e., f_θ^{QC}) and then get updated using standard policy gradient reinforcement learning algorithms, such as REINFORCE (Williams, 1992), to maximize the ranking losses of the QC model. Formally, the QC generator aims to maximize the following expected reward: $J(\phi)=\sum_i \mathbb{E}_{c^{(j)}\sim P_\phi(c|q^{(i)})}[l_\theta(q^{(i)}, c^{(i)}, c^{(j)})]$, where $l_\theta(q^{(i)}, c^{(i)}, c^{(j)})$ is the pairwise ranking loss of the discriminator model defined earlier. The gradient of J can be derived as $\nabla_\phi J=\sum_i \mathbb{E}_{c^{(j)}\sim P_\phi(c|q^{(i)})}[l_\theta \cdot \nabla_\phi \, log P_\phi(c^{(j)}|q^{(i)})]$. Another option is to let g_ϕ^{QC} use the same architecture as f_θ^{QC} and use tied parameters (i.e., ϕ=θ), as adopted in previous work (Deshpande and M.Khapra, 2019; Park and Chang, 2019).

The focus of this work is to show the effectiveness of applying adversarial learning to code retrieval, and how to regularize it with QD relevance. We leave more complex adversarial techniques (e.g. adversarial perturbation (Goodfellow et al., 2014b; Miyato et al., 2015) or adversarial sequence generation (Li et al., 2018)) for future studies.

3.2 Question-Description Relevance Regularization

Intuitively, we can train a better code retrieval model, if the negative code snippets are all true-negative ones, i.e., if they are confusingly similar to correct code answers, but perform different functionalities. However, because of the one-to-many mapping issue, some negative code snippets sampled by the adversarial QC generator can be false-negative, i.e. they are equally good answers for a given question despite that they are not paired with the question in the training set. Unfortunately during training, this problem could become increasingly obvious as the adversarial will be improved along with the code retrieval model, and eventually makes learning less and less effective. Since both the QC model and the adversarial QC generator operates from the QC perspective, it is difficult to further discriminate true-negative and false-negative code snippets.

Therefore, we propose to alleviate this problem with QD relevance regularization. This idea is inspired by the group of QC work mentioned in Section 1 that retrieves code snippets by matching their NL descriptions with a given question. But different from them, we only leverage QD relevance during training to provide a secondary view and to reweight the adversarial samples. Fortunately, an adversarial code snippet $\hat{c}$ sampled from the original training dataset $\mathcal{D}^{\text{QC}}$ is paired with an NL question $\hat{q}$, which can be regarded as its NL description and used to calculate the relevance to the given question q.

Let us refer to the example in Table 1 again. At a certain point of training, with $q^{(1)}$ "*Flatten a shallow list in Python*" being the given question, the adversarial QC generator may choose $c^{(2)}$ and $c^{(3)}$ as the negative samples, but instead of treating them equivalently, we can infer from the QD matching perspective that $c^{(3)}$ is likely to be true negative, because $q^{(3)}$ "*How to get all possible combinations of a list's elements*" clearly has different meanings from $q^{(1)}$, while $c^{(2)}$ is likely to be a false negative example since $q^{(2)}$ "*How to flatten a 2D list to 1D without using numpy?*" is similar to $q^{(1)}$. Hence, during training, the discriminative QC model should put more weights on negative samples like $c^{(3)}$ rather than $c^{(2)}$.

We now explain how to map QD relevance scores to regularization weights. Let $f^{\text{QD}}(q, \hat{q})$ denote the predicted relevance score between the given question q and the question paired with an adversarial code snippet $\hat{q}$, and let $f^{\text{QD}}(q, \hat{q})$ be normalized to the range from 0 to 1. We can see from the above example that QD relevance and adjusted learning weight should be reversely associated, so we map the normalized relevance score to a weight using a monotonously decreasing polynomial function: $w^{\text{QD}}(x)=(1-x^a)^b$, $0\leq x\leq 1$. Both a and b are positive integer hyper-parameters that control the shape of the curve and can be tuned on the dev sets. In this work, they are both set to one by default for simplicity. $w^{\text{QD}} \in [0, 1]$ allows the optimization objective to weigh less on adversarial samples that

Algorithm 1: Question-Description Relevance Regularized Adversarial Learning.

QC training data : $\mathcal{D}^{\text{QC}} = \{q^{(i)}, c^{(i)}\}$
QD model : f^{QD}
Constants : positive intergers N, τ, a, b
Result: QC model f_θ^{QC}

1 ▷ Pretrain f_θ^{QC} on $\mathcal{D}^{\text{QC}}$ using pairwise ranking loss l_θ^{QC} with randomly sampled negative code snippets ;
2 ▷ Initialize QC generator g_ϕ^{QC} with f_θ^{QC}: $\phi \leftarrow \theta$;
3 **while** *not converge or not reach max iter number* **do**
4 **for** *random sampled* $\langle q^{(i)}, c^{(i)} \rangle \in \mathcal{D}^{QC}$ **do**
5 Randomly choose D=$\{q,c\} \subset \mathcal{D}^{\text{QC}}$, where $|D|$=N;
6 Sample $c^{(j)} \in D$, that $c^{(j)} \sim P_\phi(c^{(j)}|q^{(i)}) = \text{softmax}_\tau(g_\phi^{\text{QC}}(q^{(i)}, c^{(j)}))$;
7 $l_\theta^{\text{QC}} \leftarrow l_\theta(q^{(i)}, c^{(i)}, c^{(j)})$;
8 Find $q^{(j)}$ associated with $q^{(j)}$, $w^{\text{QD}} \leftarrow (1 - f^{\text{QD}}(q^{(i)}, q^{(j)})^a)^b$;
9 Update QC model with gradient descent to reduce loss: $w^{\text{QD}} \cdot l_\theta^{\text{QC}}$;
10 Update adversarial QC generator with gradient ascent: $l_\theta^{\text{QC}} \cdot \nabla_\phi \, log P_\phi(c^{(j)}|q^{(i)})$
11 **end**
12 ▷ Optional QD model update. (See Section 3.4.)
13 **end**

are more likely to be false negative.

3.3 Question-Description Relevance Regularized Adversarial Learning

Now we describe the proposed learning framework in Algorithm 1 that combines adversarial learning and QD relevance regularization. Let us first assume the QD model is given and we will explain how to pre-train, and optionally update it shortly.

The QC model can be first pre-trained on $\mathcal{D}^{\text{QC}}$ using standard pairwise ranking loss $l_\theta(q^{(i)}, c^{(i)}, c^{(j)})$ with randomly sampled $c^{(j)}$. Line 3-11 show the QC model training steps. For each QC pair $\langle q^{(i)}, c^{(i)} \rangle$, a batch of negative QC pairs are sampled randomly from the training set $\mathcal{D}^{\text{QC}}$. The QC generator then choose an adversarial $c^{(j)}$ from distribution $P_\phi(c|q^{(i)})$ defined in Section 3.1, and its paired question is $q^{(j)}$. Two questions $q^{(i)}$ and $q^{(j)}$ are then passed to the QD model, and the QD relevance prediction is mapped to a regularization weight w^{QD}. Finally, the regularization weight is used to control the update of the QC model on the ranking loss with the adversarial $\hat{c}$.

3.4 Base Model Architecture

Our framework can be instantiated with various model architectures for QC or QD. Here we choose the same neural network architecture as (Gu et al., 2018; Yao et al., 2019) as our base QC model, that achieves competitive or state-of-the-art code retrieval performances. Concretely, both a natural language question q and a code snippet c are sequences of tokens. They are encoded respectively by separate bi-LSTM networks (Schuster and Paliwal, 1997), passed through a max pooling layer to extract the most salient features of the entire sequence, and then through a hyperbolic tangent activate function. The encoded question and code representations are denoted as h^q and h^c. Finally, a matching component scores the vector representation between q and c and outputs their matching score for ranking. We follow previous work to use cosine similarity: $f^{\text{QC}}(q, c) = \text{cosine}(h^q, h^c)$.

QD Model. There are various model architecture choices, but here for simplicity, we adapt the QC model for QD relevance prediction. We let the QD model use the same neural architecture as the QC model, but with Siamese question encoders. The QD relevance score is the cosine similarity between $h^{q^{(i)}}$ and $h^{q^{(j)}}$, the bi-LSTM encoding outputs for question $q^{(i)}$ and $q^{(j)}$ respectively: $f^{\text{QD}}(q^{(i)},q^{(j)})$=cosine$(h^{q^{(i)}},h^{q^{(j)}})$. This method allows using a pre-trained QC model to initialize the QD model parameters, which is easy to implement and the pre-trained question encoder in the QC model can help the QD performance. Since programming-domain question paraphrases are rare, we collect a small QD training set consisting of programming related natural language question pairs $\mathcal{D}^{\text{QD}}$=$\{q^{(j)}, p^{(j)}\}$ based on duplicated questions in Stack Overflow.

The learning framework can be symmetrically applied, as indicated by Line 12 in Algorithm 1, so that the QD model can also be improved. This may provide better QD relevance feedback to help train a better QC model. In short, we can use a discriminative and a generative QD model. The generative QD model selects adversarial questions to help train the discriminative QD model, and this training can be regularized by the relevance predictions from a QC model. More details will be introduced in the experiments.

4 Experiments

In this section, we first introduce our experimental setup, and then will show that our method not only outperforms the baseline methods, but also multi-task learning approaches, where question-description relevance prediction is the other task. In

	Python			SQL		
	Train	**Dev**	**Test**	**Train**	**Dev**	**Test**
QC	68,235	8,529	8,530	60,509	7,564	7,564
QD	1,085	1,085	1,447	18,020	2,252	2,253

Table 2: Dataset statistics. QD is used to represent the duplicate question dataset.

particular, the QD relevance regularization consistently improves QC performance upon adversarial learning, and the effectiveness of relevance regularization can also be verified as it is symmetrically applied to improve the QD task.

4.1 Datasets

We use StaQC (Yao et al., 2018) to train and evaluate our code retrieval model, which contains automatically extracted questions on Python and SQL and their associated code answers from Stack Overflow. We use the version of StaQC that each question is associated with a single answer, as those associated with multiple answers are predicted by an automatic answer detection model and therefore noisier. We randomly split this QC datasets by a 70/15/15 ratio into training, dev and testing sets. The dataset statistics are summarized in Table 2.

We use Stack Exchange Data Explorer[3] to collect data for training and evaluating QD relevance prediction. Specifically, we collect the question pairs from posts that are manually labeled as duplicate by users, which are related by `LinkTypeId=3`. It turns out that the QD datasets are substantially smaller than the QC datasets, especially for Python, as shown in Table 2. This makes it more interesting to check whether a small amount of QD relevance guidance can help improve code retrieval performances.

4.2 Baselines and Evaluation Metrics

We select state-of-the-art methods from both groups of work for QC (mentioned in Introduction). DecAtt and DCS below are methods that directly match questions with code. EditDist and vMF-VAE transfer code retrieval into a question matching problem.

- DecAtt (Parikh et al., 2016). This is a widely used neural network model with attention mechanism for sentence pairwise modeling.
- DCS (Gu et al., 2018). We use this as our base model, because it is a simple yet effective code retrieval model that achieves competitive performance without introducing additional training overheads (Yao et al., 2019). Its architecture has been described in Section 3.4.
- EditDist (Hayati et al., 2018). Code snippets are retrieved by measuring an edit distance based similarity function between their associated NL descriptions and the input questions. Since there is only one question for each sample in the QC datasets, we apply a standard code summarization tool (Iyer et al., 2016) to generate code descriptions to match with input questions.
- vMF-VAE (Guo et al., 2019). This is similar to EditDist, but a vMF Variational Autoencoder (Xu and Durrett, 2018) is separately trained to embed questions and code descriptions into latent vector distributions, whose distance is then measured by KL-divergence. This method is also used by Hashimoto et al. (2018).

We further consider multi-task learning (MTL) as an alternative way how QD can help QC. It is worth mentioning that our method does not require *associated* training data or the sharing of trained parameters between the QD and QC tasks, whereas MTL typically does. For fair comparison, we adapt two MTL methods to our scenario that use the same base model, or its question and code encoders:

- MTL-DCS. This is a straightfoward MTL adaptation of DCS, where the code encoder is updated on the QC dataset and the question encoder is updated on both QC and QD datasets. The model is alternatively trained on both datasets.
- MTL-MLP (Gonzalez et al., 2018). This recent MTL method is originally designed to rank relevant questions and question-related comments. It uses a multi-layer perceptron (MLP) network with one shared hidden layer, a task-specific hidden layer and a task-specific classification layer for each output. We adapt it for our task. The input to the MLP is the concatenation of similarity features $[max(h^q, h^c), h^q - h^c, h^q \odot h^c]$, where $\odot$ is element-wise product. h^q and h^c are learned using the same encoders as our base model.

The ranking metrics used for evaluation are Mean Average Precision (MAP) and Normalize Discounted Cumulative Gain (nDCG) (Järvelin and Kekäläinen, 2002). The same evaluation method as previous work is adopted (Iyer et al., 2016; Yao et al., 2019) for both QC and QD, where we randomly choose from the testing set a fixed-size (49) pool of negative candidates for each question, and

[3] SEDE and SEDE schema documentation.

	Python		SQL	
	MAP	**nDCG**	**MAP**	**nDCG**
EditDist (Hayati et al., 2018)	0.2348	0.3844	0.2096	0.3641
vMF-VAE (Guo et al., 2019)	0.2886	0.4511	0.2921	0.4537
DecAtt (Parikh et al., 2016)	0.5744	0.6716	0.5142	0.6231
DCS (Gu et al., 2018)	0.6015	0.6929	0.5155	0.6237
MTL-MLP (Gonzalez et al., 2018)	0.5737	0.6712	0.5079	0.6179
MTL-DCS	0.6024	0.6935	0.5160	0.6237
Our	**0.6372***	**0.7206***	**0.5404***	**0.6429***
Our - RR	0.6249*	0.7111*	0.5274*	0.6327*

Table 3: Code retrieval (QC) performance on test sets. * denotes significantly different from DCS (Gu et al., 2018) in one-tailed t-test ($p < 0.01$).

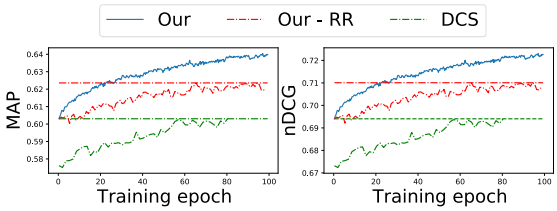


Figure 2: QC learning curves on the Python dev set.

evaluate the ranking of its paired code snippet or questions among these negative candidates.

4.3 Implementation Details

Our implementation is based on Yao et al. (2019). We follow this work to set the base model hyperparameters. The vocabulary embedding size for both natural language and programming language is set at 200. The LSTM hidden size is 400. Margin in the hinge loss is 0.05. The trained DCS model is used as pre-training for our models. The learning rate is set at 1e-4 and the dropout rate set at 0.25. For adversarial training, we set τ to 0.2 following (Wang et al., 2017) and limit the maximum number of epochs to 300. Standard L2-regularization is used on all the models. We empirically tried to tie the parameters of the discriminator and the generator following previous work (Deshpande and M.Khapra, 2019; Park and Chang, 2019), which shows similar improvements over the baselines. Implementation from Xu and Durrett (2018) is used for the vMF-VAE baseline.

We follow the code preprocessing steps in Yao et al. (2018) for Python and Iyer et al. (2016) for SQL. We use the NLTK toolkit (Bird and Loper, 2004) to tokenize the collected duplicate questions, and let it share the same NL vocabulary as the QC dataset $\mathcal{D}^{\text{QC}}$.

4.4 Results and Analyses

Our experiments aim to answer the following research questions:

(1) *Can the question regularized adversarial learning framework improve code retrieval (QC) performance?* We will first compare the code retrieval performance of different methods. Table 3 summarizes the *test results*, which are consistent on both Python and SQL datasets. Code retrieval baselines by measuring QD relevance, e.g., EditDist and vMF-VAE, are popularly used in code generation related work, but do not perform well compared to other code retrieval baselines in our experiments, partly because they are not optimized toward the QC task. This suggests that applying more advanced code retrieval methods for retrieve-and-edit code generation can be an interesting future research topic. DCS is a strong baseline, as it outperforms DecAtt that uses a more complex attention mechanism. This indicates that it is not easy to automatically learn pairwise token associations between natural language and programming languages from software community data, which is also suggested by previous work (Panthaplackel et al., 2019; Vinayakarao et al., 2017).

Our proposed learning algorithm can improve the QC performance compared to all the baselines. The "- RR" variant is to only apply adversarial sampling without QD relevance regularization. It already leads to improvements compared to the base model (i.e. DCS), but does not perform as well as our full model. This proves the usefulness of the QD relevance regularization and indicates that selectively weighting the contribution of adversarial samples to the training loss can help the model generalize better to test data. Figure 2 compares QC learning curves on the *dev set*. The full model curve being the smoothest qualitatively suggests that the adversarial learning has been well regularized.

(2) *How does the proposed algorithm compare with multi-task learning methods?* The results are reported in Table 4. The MTL-MLP model is originally proposed to improve question-question relevance prediction by using question-comment relevance prediction as a secondary task (Gonzalez et al., 2018). It does not perform as well as MTL-DCS, which basically uses hard parameter sharing between the two tasks and does not require additional similarity feature definitions. In general, the effectiveness of these MTL baselines on the QC task is limited because there are only a small amount of QD pairs available for training. Both our method and its ablated variant outperform the

	Python		SQL	
	MAP	**nDCG**	**MAP**	**nDCG**
MTL-MLP (Gonzalez et al., 2018)	0.5737	0.6712	0.5079	0.6179
MTL-DCS	0.6024	0.6935	0.5160	0.6237
Our	**0.6372**	**0.7206**	**0.5404**	**0.6429**

Table 4: Compare QC performance with MTL.

MTL baselines. This shows that it may be more effective to use a data scarce task to regularize the adversarial learning of a relatively data rich task, than using those scarce data in MTL.

(3) *Can the QD performance be improved by the proposed method?* Although QD is not the focus of this work, we can use it to verify that generalizability of our method by symmetrically applying it to update the QD model as mentioned in Section 3.2. To be concrete, a generative adversarial QD model selects difficult questions from the a distribution of question pair scores: $\hat{q} \sim \text{softmax}_\tau(f^{\text{QD}}(\hat{q}, q^{(i)}))$. Then a QC model is used to calculate a relevance score for a question-code pair, and this can regularize the adversarial learning of the QD model.

Table 5 shows the results. Our method and its ablated variants outperform the QD baselines EditDist and vMF-VAE, again suggesting that supervised learning is more effective. The full model achieves the best overall performance and removing relevance regularization (- RR) from the QC model consistently leads to performance drop. In contrast, further removing adversarial sampling (- AS) hurts the performance on SQL dataset slightly, but not on Python. This is probably because the Python QD dataset is very small and using adversarial learning can easily overfit, which again suggests the importance of our proposed relevance regularization. Finally, removing QC as pretraining (- Pretrain) greatly hurts the performance, which is understandable since QC datasets are much larger.

Because the QD model performance can be improved in such a way, we allow it to get updated in our QC experiments (corresponding to line 12 in Algorithm 1) and the results have been discussed in Table 3. We report here the QC performance using a fixed QD model (i.e. Our - RR - AS) for relevance regularization: MAP=0.6371, nDCG=0.7205 for Python and MAP=0.5366, nDCG=0.6398 for SQL. Comparing these results with those in Table3 (Our), one can see that allowing the QD model to update consistently improves QC performance, which suggests that a better QD model can provide more accurate relevance regularization to the QC model and leads to better results.

	Python		SQL	
	MAP	**nDCG**	**MAP**	**nDCG**
EditDist (Hayati et al., 2018)	0.3617	0.4883	0.3246	0.4580
vMF-VAE (Guo et al., 2019)	0.3009	0.4616	0.3029	0.4641
Our	**0.7162**	**0.7821**	**0.6947**	**0.7651**
Our - RR	0.7046	0.7734	0.6846	0.7575
Our - RR - AS	0.7116	0.7787	0.6764	0.7512
Our - RR - AS - Pretrain	0.3882	0.5170	0.6284	0.7129

Table 5: Question relevance prediction results, evaluated on the question duplication dataset we collected.

5 Related Work

Code Retrieval. Code retrieval has developed from using classic information retrieval techniques (Hill et al., 2014; Haiduc et al., 2013; Lu et al., 2015) to recently deep neural methods that can be categorized into two groups. The first group directly model the similarity across the natural language and programming language modalities. Besides CODENN (Iyer et al., 2016) and DCS (Gu et al., 2018) discussed earlier, Yao et al. (2019) leverage an extra code summarization task and ensemble a separately trained code summary retrieval model with a QC model to achieve better overall code retrieval performances. Ye et al. (2020) further train a code generation model and a code summarization model through dual learning, which helped to learn better NL question and code representations. Both works employ additional sequence generation models that greatly increases the training complexity, and they both treat all unpaired code equally as negatives. Our work differs from them as we introduce adversarial learning for code retrieval, and the existing work do not leverage question relevance for code retrieval as we do.

The second group of works transfer code retrieve to a code description retrieval problem, similar to general domain question answering, where two natural language sentences are matched (Zhao et al., 2017, 2019b). This methodology has been widely adopted as a component in the retrieve-and-edit code generation literature. For example, heuristic methods such as measuring edit distance (Hayati et al., 2018) or comparing code type and length (Huang et al., 2018) are used, and separate question latent representations (Hayati et al., 2018; Guo et al., 2019) are learned. Our work shares with them the idea to exploit QD relevance, but we use QD relevance in a novel way to regularize the adversarial learning of QC models. It will be an interesting future work to leverage the proposed code retrieval method for retrieve-and-edit code generation.

Adversarial Learning. Adversarial learning has been widely used in areas such as computer vision (Mirza and Osindero, 2014; Chen et al., 2016; Radford et al., 2015; Arjovsky et al., 2017), text generation (Yu et al., 2017; Chen et al., 2019; Liang, 2019; Gu et al., 2018; Liu et al., 2017; Ma et al., 2019; Zhao et al., 2019a), relation extraction (Wu et al., 2017; Qin et al., 2018), question answering (Oh et al., 2019; Yang et al., 2019), etc. We proposed to apply adversarial learning to code retrieval, because they have effectively improved cross-domain task performances and helped generate useful training data, We adapted the method from Wang et al. (2017) for the bi-modal QC scenario. As future work, adversarial learning for QC can be generalized to other settings with different base neural models (Yang et al., 2019) or with more complex adversarial learning methods, such as adding perturbed noises (Park and Chang, 2019) or generating adversarial sequences (Yu et al., 2017; Li et al., 2018). Our method differs from most adversarial learning work in that the discriminator (QC model) does not see all generated samples as equally negative.

6 Conclusion

This work studies the code retrieval problem, and tries to tackle the challenges of matching natural language questions with programming language (code) snippets. We propose a novel learning algorithm that introduces adversarial learning to code retrieval, and it is further regularized from the perspective of a question-description relevance prediction model. Empirical results show that the proposed method can significantly improve the code retrieval performances on large-scale datasets for both Python and SQL programming languages.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, and NSF CAREER #1942980. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Rajas Agashe, Srini Iyer, and Luke Zettlemoyer. 2019. Juice: A large scale distantly supervised dataset for open domain context-based code generation. *ArXiv*, abs/1910.02216.

Shayan A. Akbar and Avinash C. Kak. 2019. Scor: Source code retrieval with semantics and order. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 1–12.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *ICML*.

Bin Bi, Chen Wu, Ming Yan, Wei Wang, Jiangnan Xia, and Chenliang Li. 2019. Incorporating external knowledge into machine reading for generative question answering. *ArXiv*, abs/1909.02745.

Steven Bird and Edward Loper. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain. Association for Computational Linguistics.

MK Chen, Xinyi Lin, Chen Wei, and Rui Yan. 2019. Bofgan: Towards a new structure of backward-or-forward generative adversarial nets. In *The World Wide Web Conference*, pages 2652–2658. ACM.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*.

Zimin Chen and Martin Monperrus. 2019. A literature study of embeddings on source code. *ArXiv*, abs/1904.03061.

Milan Cvitkovic, Badal Singh, and Anima Anandkumar. 2019. Open vocabulary learning on source code with a graph-structured cache. In *ICML*.

Ameet Deshpande and Mitesh M.Khapra. 2019. Dissecting an adversarial framework for information retrieval.

Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. 2020. Hoppity: Learning graph transformations to detect and fix bugs in programs. In *ICLR*.

Ana Gonzalez, Isabelle Augenstein, and Anders Søgaard. 2018. A strong baseline for question relevancy ranking. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4810–4815.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014a. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014b. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 933–944. IEEE.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2019. Coupling retrieval and meta-learning for context-dependent semantic parsing. In *ACL*.

Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 842–851. IEEE Press.

Tatsunori B. Hashimoto, Kelvin Guu, Yonatan Oren, and Percy Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. *ArXiv*, abs/1812.01194.

Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. 2018. Retrieval-based neural code generation. In *EMNLP*.

Emily Hill, Manuel Roldan-Vega, Jerry Alan Fails, and Greg Mallet. 2014. Nl-based query refinement and contextualized code search results: A user study. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 34–43. IEEE.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *NAACL-HLT*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083.

Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

Dong-Jin Kim, Jinsoo Choi, Tae-Hyun Oh, and In So Kweon. 2019. Image captioning with very scarce supervised data: Adversarial semi-supervised learning approach. In *EMNLP/IJCNLP*.

Alexander LeClair and Collin McMillan. 2019. Recommendations for datasets for source code summarization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3931–3937, Minneapolis, Minnesota. Association for Computational Linguistics.

Dianqi Li, Qiuyuan Huang, Xiaodong He, Lei Zhang, and Ming-Ting Sun. 2018. Generating diverse and accurate visual captions by comparative adversarial learning. *ArXiv*, abs/1804.00861.

Shangsong Liang. 2019. Unsupervised semantic generative adversarial networks for expert retrieval. In *The World Wide Web Conference*, pages 1039–1050. ACM.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial multi-task learning for text classification. In *ACL*.

Meili Lu, Xiaobing Sun, Shaowei Wang, David Lo, and Yucong Duan. 2015. Query expansion via wordnet for effective code search. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 545–549. IEEE.

Jing Ma, Wei Gao, and Kam-Fai Wong. 2019. Detect rumors on twitter by promoting information campaigns with generative adversarial learning. In *The World Wide Web Conference*, pages 3049–3055. ACM.

Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *ArXiv*, abs/1411.1784.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. 2015. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*.

Jong-Hoon Oh, Kazuma Kadowaki, Julien Kloetzer, Ryu Iida, and Kentaro Torisawa. 2019. Open-domain why-question answering with adversarial learning to encode answer texts. In *ACL*.

Sheena Panthaplackel, Milos Gligoric, Raymond J. Mooney, and Junyi Jessy Li. 2019. Associating natural language comment and source code entities. *ArXiv*, abs/1912.06728.

Sheena Panthaplackel, Pengyu Nie, Milos Gligoric, Junyi Jessy Li, and Raymond J. Mooney. 2020. Learning to update natural language comments based on code changes. *ArXiv*, abs/2004.12169.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas. Association for Computational Linguistics.

Dae Hoon Park and Yi Chang. 2019. Adversarial sampling and training for semi-supervised information retrieval. In *The World Wide Web Conference*, pages 1443–1453. ACM.

Pengda Qin, Weiran Xu, and William Yang Wang. 2018. Dsgan: Generative adversarial training for distant supervision relation extraction. In *ACL*.

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. 2017. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7167–7176.

Venkatesh Vinayakarao, Anita Sarma, Rahul Purandare, Shuktika Jain, and Saumya Jain. 2017. Anne: Improving source code search using entity retrieval approach. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 211–220. ACM.

Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 515–524. ACM.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Yi Wu, David Bamman, and Stuart J. Russell. 2017. Adversarial training for relation extraction. In *EMNLP*.

Jiacheng Xu and Greg Durrett. 2018. Spherical latent spaces for stable variational autoencoders. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Xiao Yang, Madian Khabsa, Miaosen Wang, Wei Wang, Ahmed Hassan Awadallah, Daniel Kifer, and C Lee Giles. 2019. Adversarial training for community question answer selection based on multi-scale matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 395–402.

Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. 2019. Coacor: Code annotation for code retrieval with reinforcement learning. In *The World Wide Web Conference*, pages 2203–2214. ACM.

Ziyu Yao, Daniel S Weld, Wei-Peng Chen, and Huan Sun. 2018. Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference*, pages 1693–1703. International World Wide Web Conferences Steering Committee.

Wei Ye, Rui Xie, Jinglei Zhang, Tianxiang Hu, Xiaoyin Wang, and Shikun Zhang. 2020. Leveraging code generation to improve code retrieval and summarization via dual learning. In *Proceedings of The Web Conference 2020*, pages 2309–2319.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Jie Zhao, Xiang Deng, and Huan Sun. 2019a. Easy-to-hard: Leveraging simple questions for complex question generation. *arXiv preprint arXiv:1912.02367*.

Jie Zhao, Ziyu Guan, and Huan Sun. 2019b. Riker: Mining rich keyword representations for interpretable product question answering. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1389–1398.

Jie Zhao, Yu Su, Ziyu Guan, and Huan Sun. 2017. An end-to-end deep framework for answer triggering with a novel group-level objective. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1276–1282.

Clinical Reading Comprehension: A Thorough Analysis of the emrQA Dataset

Xiang Yue **Bernal Jimenez Gutierrez** **Huan Sun**
The Ohio State University
{yue.149, jimenezgutierrez.1, sun.397}@osu.edu

Abstract

Machine reading comprehension has made great progress in recent years owing to large-scale annotated datasets. In the clinical domain, however, creating such datasets is quite difficult due to the domain expertise required for annotation. Recently, Pampari et al. (2018) tackled this issue by using expert-annotated question templates and existing i2b2 annotations to create *emrQA*, the first large-scale dataset for question answering (QA) based on clinical notes. In this paper, we provide an in-depth analysis of this dataset and the clinical reading comprehension (CliniRC) task. From our qualitative analysis, we find that (i) emrQA answers are often incomplete, and (ii) emrQA questions are often answerable without using domain knowledge. From our quantitative experiments, surprising results include that (iii) using a small sampled subset (5%-20%), we can obtain roughly equal performance compared to the model trained on the entire dataset, (iv) this performance is close to human expert's performance, and (v) BERT models do not beat the best performing base model. Following our analysis of the emrQA, we further explore two desired aspects of CliniRC systems: the ability to utilize clinical domain knowledge and to generalize to unseen questions and contexts. We argue that both should be considered when creating future datasets.[1]

1 Introduction

Medical professionals often query over clinical notes in Electronic Medical Records (EMRs) to find information that can support their decision making (Demner-Fushman et al., 2009; Rosenbloom et al., 2011; Wang et al., 2018). One way to facilitate such information seeking activities is to build a natural language question answering (QA) system that can extract precise answers from clinical notes (Cairns et al., 2011; Cao et al., 2011; Wren, 2011; Abacha and Demner-Fushman, 2016, 2019).

[1]Our code is available at https://github.com/xiangyue9607/CliniRC.

RECORD #992321, Date: 2145-09-22

Context: ... **For HTN control, pt was given HCTZ** and lopressor which sufficiently controlled his BP. Pt was sent home on HCTZ 25mg daily and atenolol 50mg daily.
...
ADDITIONAL COMMENTS: 1.) **Take hydrochlorothiazide 25mg daily** and atenolol 50mg daily for your blood pressure. You should also take aspirin 81mg daily.

Question: Why has the patient been prescribed hctz?
Answer: *For HTN control, pt was given HCTZ and lopressor which sufficiently*

Question: What was the dosage prescribed of hydrochlorothiazide?
Answer: *ADDITIONAL COMMENTS: 1.) Take hydrochlorothiazide 25mg daily and atenolol 50mg daily for your*

Figure 1: Examples from the emrQA dataset: Part of a clinical note as *context* and 2 *question-answer* pairs. Due to the original emrQA generation issues, oftentimes answers are incomplete or contain irrelevant parts to the questions (the underlined parts are what we think the most relevant to the questions).

Machine reading comprehension (RC) aims to automatically answer questions based on a given document or text corpus and has drawn wide attention in recent years. Many neural models (Cheng et al., 2016; Wang et al., 2017; Wang and Jiang, 2017; Seo et al., 2017; Chen et al., 2017; Devlin et al., 2019) have achieved very promising results on this task, owing to large-scale QA datasets (Hermann et al., 2015; Rajpurkar et al., 2016; Trischler et al., 2017; Joshi et al., 2017; Yang et al., 2018). Unfortunately, clinical reading comprehension (CliniRC) has not observed as much progress due to the lack of such QA datasets.

In order to create QA pairs on clinical texts, annotators must have considerable medical expertise and data handling must be specifically designed to address ethical issues and privacy concerns. Due to these requirements, using crowdsourcing like in the open domain to create large-scale clinical QA

datasets becomes highly impractical (Wei et al., 2018).

Recently, Pampari et al. (2018) found a smart way to tackle this issue and created *emrQA*, the first large-scale QA dataset on clinical texts. Instead of relying on crowdsourcing, emrQA was semi-automatically generated based on annotated question templates and existing annotations from the n2c2 (previously called i2b2) challenge datasets[2]. Example QA pairs from the dataset are shown in Figure 1.

In this paper, we aim to gain a deep understanding of the CliniRC task and conduct a thorough analysis of the emrQA dataset. We first explore the dataset directly by carrying out a meticulous qualitative analysis on randomly-sampled QA pairs and we find that: 1) Many answers in the emrQA dataset are incomplete and hence are hard to read and ineffective for training (§3.1). 2) Many questions are simple: More than 96% of the examples contain the same key phrases in both questions and answers. Though Pampari et al. (2018) claims that 39% of the questions may need knowledge to answer, our error analysis suggests only a very small portion of the errors (2%) made by a state-of-the-art reader might be due to missing external domain knowledge (§3.2).

Following our qualitative analysis of the emrQA dataset, we conduct a comprehensive quantitative analysis based on state-of-the-art readers and BERT models (BERT-base (Devlin et al., 2019) as well as its biomedical and clinical versions: BioBERT (Lee et al., 2019) and ClinicalBERT (Alsentzer et al., 2019)) to understand how different systems behave on the emrQA dataset. Surprising results include: 1) Using a small sampled subset (5%-20%), we can obtain roughly equal performance compared to the model trained on the entire dataset, suggesting that many examples in the dataset are redundant (§4.1). 2) The performance of the best base model is close to the human expert's performance[3] (§4.2). 3) The performance of BERT models is around 1%-5% worse than the best performing base model (§4.3).

After completing our analysis of the dataset, we explore two potential needs for systems doing CliniRC: 1) The need to represent and use clinical domain knowledge effectively (§5.1) and 2) the need to generalize to unseen questions and contexts (§5.2). To investigate the first one, we analyze several types of clinical questions that require domain knowledge and can frequently appear in the real clinical setting. We also carry out an experiment showing that adding knowledge explicitly yields around 5% increase in F1 over the base model when tested on samples that we created by altering the original questions to involve semantic relations. To study generalizability, we ask medical experts to create new questions based on the unseen clinical notes from MIMIC-III (Johnson et al., 2016), a freely accessible critical care database. We find that the performance of the best model trained on emrQA drops by 40% under this new setting, showing how critical it is for us to develop more robust and generalizable models for the CliniRC task.

In summary, given our analysis of the emrQA dataset and the task in general, we conclude that future work still needs to create better datasets to advance CliniRC. Such datasets should be not only large-scale, but also less noisy, more diverse, and allow researchers to directly evaluate a system's ability to encode domain knowledge and to generalize to new questions and contexts.

[2]https://portal.dbmi.hms.harvard.edu/projects/n2c2-nlp/

[3]Which is obtained by comparing emrQA answers to answers created by our medical experts on sampled QA pairs.

	Medication	**Relation**
# Question	222,957	904,592
# Context	261	423
# Question Template	80	139
Question: avg. tokens	8.00	7.91
Answers: avg. tokens	9.47	10.41
Context: avg. tokens	1062.66	889.23

Table 1: Statistics of two major subsets, *Medication* and *Relation*, of the emrQA dataset.

2 Overview of the emrQA dataset

Similar to the open-domain reading comprehension task, the Clinical Reading Comprehension (CliniRC) task is defined as follows:

Definition 2.1. Given a patient's clinical note (context) $C = \{c_1, ..., c_n\}$ and a question $Q = \{t_1, ..., t_m\}$, the CliniRC task aims to extract a continuous span $A = \{c_i, c_{i+1}, ..., c_{i+k}\} (1 \leq i \leq i + k \leq n)$ from the context as the answer, where c_i, t_j are tokens.

The emrQA dataset (Pampari et al., 2018) was semi-automatically generated from expert-annotated question templates and existing i2b2 annotations. More specifically, clinical question templates were first created by human experts. Then, manual annotations from the medication information extraction, relation learning, and coreference

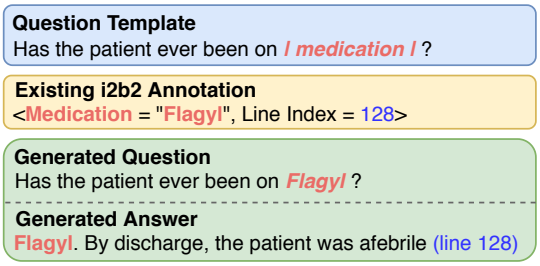


Figure 2: An example to illustrate how emrQA generates QA pairs.

resolution i2b2 challenges were re-framed into answers for the question templates. After linking question templates to i2b2 annotations, the gold annotation entities were used to both replace placeholders in the question templates and extract the sentence around them as answers. An example of this generation process can be seen in Figure 2.

The emrQA dataset contains 5 subsets: *Medication*, *Relation*, *Heart Disease*, *Obesity* and *Smoking*, which were generated from 5 i2b2 challenge datasets respectively. The answer format in each dataset is different. For the *Obesity* and *Smoking* datasets, answers are categorized into 7 classes and the task is to predict the question's class based on the context. For the *Medication*, *Relation*, and *Heart Disease* datasets, answers are usually short snippets from the text accompanied by a longer span around it which we refer to as an evidence. The short snippet is a single entity or multiple entities while the evidence contains the entire line around those entities in the clinical note. For questions that cannot be answered via entities, only the evidence is provided as an answer. Given that some questions do not have short answers and that entire evidence spans are usually important for supporting clinical decision making (Demner-Fushman et al., 2009), we treat the *answer evidence* [4] as our answer just as is done in (Pampari et al., 2018).

In this work, we mainly focus on the *Medication* and *Relation* datasets because (1) they make up 80% of the entire emrQA dataset and (2) their format is consistent with the span extraction task, which is more challenging and meaningful for clinical decision making support. We filter the answers whose lengths (number of tokens) are more than 20. The detailed statistics of the two datasets are shown in Table 1.

[4]For simplicity, we use "answer" directly henceforth.

Metric	**Medication**	**Relation**
Quality Score	3.92	4.75
EM	26.0	92.0
F1	74.7	95.4

Table 2: An estimate of the quality of answers in the *Medication* and *Relation* datasets based on the analysis of our randomly sampled 50 questions for each dataset. Quality scores are the average of two human annotators' (maximum: 5). EM and F1 scores are calculated between *human-labeled* answers v.s. *emrQA* answers.

3 In-depth Qualitative Analysis

In this section, we carry out an in-depth analysis of the emrQA dataset. We aim to examine (1) the quality and (2) level of difficulty for the generated QA pairs in the emrQA dataset.

3.1 How clean are the emrQA answers?

Since the emrQA dataset was created via a generation framework unlike human-labeled or crowdsourcing datasets, the quality of the datasets remains largely unknown. In order to use this dataset to explore the CliniRC task, it is essential to determine whether it is meaningful.

In order to do this, we randomly sample 50 QA pairs from the *Medication* and the *Relation* datasets respectively. Since some questions share the same answer due to automatic generation, we make sure all the samples have different answers.

Since the questions were generated from expert created templates, most of them are human-readable and unambiguous. We therefore mainly focus on evaluating answer quality. We ask two human experts to score each answer from 1 to 5 depending on the relevance of the answer to the question (1: irrelevant or incorrect; 2: missing key parts; 3: contains key parts but is not human-readable or contains many irrelevant parts; 4: contains key parts and is only missing a few parts or has a few irrelevant extra segments; 5: perfect answer). We also ask human annotators to label the gold answers and then calculate the Exact Match (EM) and F1 score (F1) of the emrQA answers v.s. human gold answers. The answer quality score, EM and F1 in both datasets, are shown in Table 2.

The scores of the *Medication* dataset are low since most of the answers are broken sentences or contain unnecessary segments. For instance, in the Figure 2 example, the correct answer should be *"Clindamycin was changed to Flagyl"*, how-

Error Type	Question	emrQA Answers	Prediction	Error Ratio	
				Medication	Relation
Span mismatch - include key info	Does she have a history of known drug allergies?	ALLERGIES: He had no known drug allergies	He had no known drug allergies	**78%**	**66%**
Span mismatch - miss key info	What is the current dose of lasix?	MEDS: K-Dur 20 BID, Nexium 20, lasix 160 BID	BID	**4%**	**0%**
Ambigious questions	What is the patient's low history?	At the time of discharge, her potassium had been low despite repletion	11) Low grade,anemia	**8%**	**4%**
Incorrect golds	What is the patient's incisions status?	Wash incisions with warm water and gentle soap	Do not apply lotions, creams, ointments or powders to incision	**2%**	**2%**
False negatives	Is there a mention of fluid in the record?	There is some fluid, or mucosal thickening in the ethmoid and sphenoid sinuses	The amount of fluid layering at the apices and the pleural spaces appear slightly decreased	**2%**	**18%**
May need external knowledge	What treatment has the patient had for his CAD?	CAD s/p CABG 2003 s/p	Pt's vancomycin was stopped after 14 days of treatment	**2%**	**2%**
Others	Is the patient's right hand ganglion cyst well-controlled?	right hand ganglion cyst removed	x 3 right hand ganglion cyst	**4%**	**8%**

Table 3: Error analysis on 50 sampled questions from the *Medication* and *Relation* dev sets respectively. Example question, ground truth and prediction from either *Medication* or *Relation* are given for each type of error.

ever, the emrQA answer misses important parts *"Clindamycin was changed to"* and contains irrelevant parts *"By discharge, the patient was afebrile"*. These issues are common in the *Medication* dataset and make it difficult to train a good system. To understand why the generated answers contain such noise, we explored the *"i2b2 2009 Medication"* challenge dataset which was used to create these QA pairs. We found that most documents in this dataset contain many complete sentences split into separate lines. Since the i2b2 annotation are token based and the emrQA obtains full lines around the token as evidence spans, these lines often end up being broken sentences. We tried to relabel the answers with existing sentence segmentation tools and heuristic measures but found that it is very challenging to obtain concise and complete text spans as answers.

Compared with the *Medication* dataset, the answer quality of the *Relation* dataset is much better. In most cases, the answers are complete and meaningful sentences with no unnecessary parts.

3.2 How challenging are the emrQA pairs?

Another observation from the 50 samples is that 96% of the answers in the *Medication* dataset and 100% of the answers in the *Relation* dataset contain the key phrase in the question. This is due to the generation procedure illustrated in Figure 2. In this example, the key phrase or entity (*"Flagyl"*) in the question is also included in the answer. This undoubtedly makes the answer easier to extract as long as the model can recognize significant words and do "word matching".

To further explore how much clinical language understanding is needed and what kind of errors do the state-of-the-art reader make, we conduct error analysis using DocReader (Chen et al., 2017) (also used in (Pampari et al., 2018)) on the emrQA dataset. More specifically, we randomly sample 50 questions that are answered incorrectly by the model (based on exact match metric) from the *Medication* and *Relation* dev set respectively[5]. The results are shown in Table 3 (examples for each error type are also given for better understanding).

Since emrQA answers are often incomplete in the dataset, we deem *span mismatch* errors acceptable as long as the predictions include the key part of the ground truths. Surprisingly, *span mismatch-include key info* errors, along with *ambiguous questions*, *incorrect golds* and *false negatives* (the prediction is correct but it is not in the emrQA answers) errors, which are caused by the dataset itself, account for 90% of total errors, suggesting that the accuracy of these models is even higher than we report.

[5]Note that these 100 samples are sampled from errors, which are different from the previously sampled ones.

Another interesting finding from the error analysis is that to our surprise, only a very small amount (2%) of errors may have been caused by a lack of external domain knowledge while Pampari et al. (2018) claim that 39% of the questions in the emrQA dataset need domain knowledge. This surprising result might be due to: (1) neural models being able to encode relational or associative knowledge from the text corpora as has also been reported in recent studies (Petroni et al., 2019; Bouraoui et al., 2020), and (2) questions and answers sharing key phrases (as we mentioned earlier in §3.1) in many samples, making it more likely that fewer questions need external knowledge to be answered than previously reported.

4 Comprehensive Quantitative Analysis

In this section, we conduct comprehensive experiments on the emrQA dataset with state-of-the-art readers and recently dominating BERT models. Full experimental settings are described in Appendix A.

4.1 How redundant are the emrQA pairs?

Though there are more than 1 million questions in the emrQA dataset (as shown in Table 1), many questions and their patterns are very similar since they are generated from the same question templates. This observation leads to a natural question: *do we really need so many questions to train an CliniRC system?* If many questions are similar to each other, it is very likely that using a sampled subset can achieve roughly the same performance that is based on the entire dataset.

To verify our hypothesis, we first split the two datasets into train, dev, and test set with the proportion of 7:1:2 w.r.t. the contexts (full statistics are shown in Appendix Table A1). Then we randomly sample {5%, 10%, 20%, 40%, 60%} and {1%, 3%, 5%, 10%, 15%}[6] of the QA pairs in each document (context) of the *Medication* and the *Relation* training sets respectively. We run DocReader (Chen et al., 2017) on the sampled subsets and evaluate them on the same dev and test set.

As shown in Figure 3, using 20% of the questions in the *Medication* and 5% of the questions in the *Relation* dataset can achieve roughly the same performance as using the entire training sets.

[6]The sampling percentage of the *Relation* dataset is smaller than the *Medication* dataset since the former one has more QA pairs (roughly 4 times).

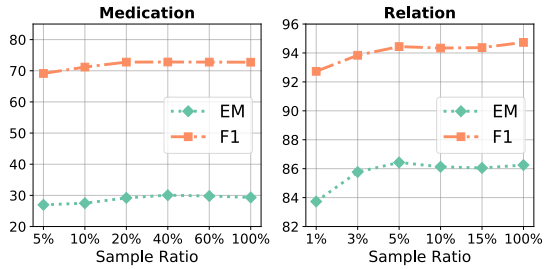


Figure 3: Impact of *training size* on the performance of DocReader (Chen et al., 2017) based on the *Medication* and *Relation* dataset.

These verify our hypothesis, and illustrate learning a good and robust reader system based on the emrQA dataset does not need so many question-answer pairs. While deep models are often data-hungry, it does not mean more data can always lead to better performance. In addition to the training size, diversity should also be considered as another important criterion for data quality.

In the following experiments, we use the sampled subsets (20% for *Medication* and 5% for *Relation*) considering the time and memory cost as well as performance.

4.2 Little room for improvement

Since the answers in emrQA are often incomplete, the performance of a model is more appropriately reflected by its F1 score. As shown in Table 2, we obtain F1 scores of 74% and 95% on two datasets respectively when we test human-labeled answers against the emrQA answers on a sampled dataset. We can see from Table 4 that the best performing reader, DocReader, achieves around 70% and 94% F1 performance on the *Medication* and *Relation* test set respectively, which are very close to the human performance just described. Though designing more complex and advanced models may achieve better scores, such scores are obtained w.r.t. noisy emrQA answers and may not translate meaningfully to real cases.

4.3 BERT does not always win

BERT models have achieved very promising results recently in various NLP tasks including RC (Devlin et al., 2019). We follow their experiment setting of BERT for doing reading comprehension on the SQuAD (Rajpurkar et al., 2016) dataset. To our surprise, as shown in Table 4, BERT models (BERT-base, its biomedical version BioBERT (Lee et al., 2019), and its clinical version ClinicalBERT

Model	Medication				Relation			
	Dev		Test		Dev		Test	
	EM	F1	EM	F1	EM	F1	EM	F1
BiDAF (Seo et al., 2017)	25.50	68.13	23.35	67.18	81.51	90.84	82.74	91.27
DocReader (Chen et al., 2017)	29.20	72.78	25.68	70.45	86.43	94.44	86.94	94.85
QANet (Yu et al., 2018)	27.67	69.40	24.74	67.34	82.41	90.61	82.68	91.56
BERT-base (Devlin et al., 2019)	26.62	68.75	24.00	67.49	80.17	90.01	83.29	92.38
BioBERT (Lee et al., 2019)	27.81	71.90	24.75	69.97	81.57	91.38	83.61	92.62
ClinicalBERT (Alsentzer et al., 2019)	27.14	71.84	24.06	69.05	83.12	91.96	85.33	93.06

Table 4: Overall performance of all models on the *Medication* and *Relation* dataset. All numbers are percentages.

(Alsentzer et al., 2019)) do not dominate as they do in the open-domain RC tasks. The reasons may be three-fold: 1) BERT benefits the most from large training corpora. The training corpora of BERT-base and BioBERT are Wikipedia + BookCorpus (Zhu et al., 2015) and PubMed articles respectively, both of which may have different vocabularies and use different language expressions from clinical texts. Though ClinicalBERT was pretrained on MIMIC-III (Johnson et al., 2016) clinical texts, the training size of the corpus (∼50M words) is far less than that used in BERT (∼3300M words), which may make the model less powerful as it is on the open-domain tasks. 2) Longer Contexts. As can be seen from Table 1, the number of tokens in the contexts is commonly larger than open-domain RC datasets like SQuAD (∼1000 v.s.∼116 avg). We suspect that long contexts might make it more challenging to model sequential information. For sequences that are longer than the *max length* of the BERT model, they are truncated into a set of short sequences, which may hinder the model from capturing long dependencies (Dai et al., 2019) and global information in the entire document. 3) Easy Questions. Another possible reason might be the question patterns are too easy and a simpler reader with far less parameters can learn the patterns and obtain satisfying performance.

Additionally, to further evaluate the models in the fine-grained level, inspired by (Gururangan et al., 2018), we partition the *Medication* and *Relation* test sets into Easy and Hard subsets using a base model. The details of Easy/Hard splits can be found in Appendix C. As can be seen from Table A4, most of the questions in the two datasets are easy, which indicates the emrQA dataset might not be challenging for the current QA models. More difficult datasets are needed to advance the Clinical Reading Comprehension task.

5 Desiderata in Real-World CliniRC

Following our analysis of the emrQA dataset, we further study two aspects of clinical reading comprehension systems that we believe are crucial for their real-world applicability: the need to encode clinical domain knowledge and to generalize to unseen questions and documents.

5.1 External domain knowledge is needed

So far, we have shown that domain knowledge may not be very useful for models answering questions *in the emrQA dataset*; however, we argue that systems in real-world CliniRC need to be able to encode and use clinical domain knowledge effectively.

Clinical text often contains high variability in many domain-specific words due to abbreviations and synonyms. The presence of different aliases in the question and context can make it difficult for a model to represent semantics accurately and choose the correct span. Besides, medical domain-specific relations (e.g., *treats, caused by*) and hierarchical relations (e.g., *isa*) between medical concepts would be likely to appear. The process followed to generate the current emrQA dataset leads to these problems being largely under-represented, even though they can be very common in real cases. We use the following 3 examples as representatives to illustrate the real cases we may encounter.

Synonym. For example, for the question in Figure 2, *"Has this patient ever been on Flagyl?"*, it is easy for the model to answer since *"Flagyl"* appears in the context. However, if we change *"Flagyl"* to its synonyms *"Metronidazole"* (which may not appear in training) in the question, it is hard for the reader to extract the correct answer, as it is not possible for model to capture the semantic meaning of *"Metronidazole"* as *"Flagyl"*.

Clinical Relations. Another example is the ques-

tion shown in Figure 1, *"Why has the patient been prescribed hctz?"*. Currently, machines can easily find the answer since keyword *"hctz"* is mentioned in the answer. However, given a situation where the drug *"hctz"* does not appear in the local context of *"HTN"*, our model may have a better chance to extract the correct answers if it stores the relation *"(hctz, treats, HTN)"*.

Hierarchical Relation. For the question *"Is there a history of mental illness?"*, it is more likely that the medical report describes a specific type of psychological condition rather than mention the general phrase *"mental illness"* since clinical support require specifics. To obtain the correct answer in this case *"Depression with previous suicidal ideation."*, encoding the relation *"(depression, isa, mental illness)"* would probably help the model make a correct prediction.

These three cases help illustrate how complex medical relations affect the real CliniRC task. Without leveraging external domain knowledge, it is difficult for models to capture the semantic relations necessary to resolve such cases.

In order to verify our claim quantitatively, we select *synonym* as a representative relation type and manipulate each question by replacing its entities with plausible synonyms or abbreviations. We then introduce external domain knowledge into current models and compare their performance against base models on these augmented questions.

More specifically, we first detect entities in the questions and link them to a medical knowledge base (KB): UMLS (Bodenreider, 2004) using a biomedical and clinical text NLP pipeline tool, *ScispaCy* (Neumann et al., 2019). Synonyms of detected entities are then retrieved from UMLS and used to replace the original mention. We filter the questions that do not contain entities or that contain entities with no synonyms. We focus on the *Relation* dataset and only modify the questions in the dev and test set; the questions in the training set are not modified. Finally, we get 69,912 and 125,338 questions in the dev and test set.

We then introduce a simple Knowledge Incorporation Module (KIM) to evaluate the usefulness of external domain knowledge. Formally, given a question $q : \{w_1^q, w_2^q, ..., w_l^q\}$ and its context $c : \{w_1^c, w_2^c, ..., w_m^c\}$, where w_i^q, w_j^c are words (tokens), all the words can be mapped to d_1 dimensional vectors via a word embedding matrix $E_w \in \mathbb{R}^{d_1 \times |\mathcal{V}|}$, where $\mathcal{V}$ denotes the word vocabulary. So we have $q : \mathbf{w_1^q}, ..., \mathbf{w_l^q} \in \mathbb{R}^{d_1}$ and $c : \mathbf{w_1^c}, ..., \mathbf{w_m^c} \in \mathbb{R}^{d_1}$.

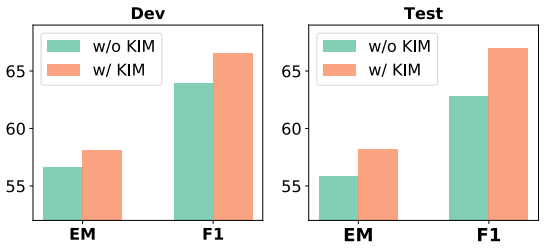


Figure 4: Performances of DocReader and DocReader + Knowledge Incorporation Module (KIM) on our created questions modified from the *Relation* dataset.

We then detect entities $\{e_1^q, e_2^q, ..., e_n^q\}$ in the question and entities $\{e_1^c, e_2^c, ..., e_o^q\}$ in the context and map them to a medical knowledge base (KB), UMLS (Bodenreider, 2004) using scispacy (Neumann et al., 2019). Note that l is not equal to n and m is not equal to o, since not every token can be mapped to a entity in KB. For entities that contain multiple words, we align them to the first token, same as the alignment used in (Zhang et al., 2019). We then map detected entities to d_2 dimensional vectors $\{\mathbf{e_1^q}, \mathbf{e_2^q}, ..., \mathbf{e_n^q}\}$ and $\{\mathbf{e_1^c}, \mathbf{e_2^c}, ..., \mathbf{e_o^c}\}$ via a entity embedding matrix $E_e \in \mathbb{R}^{d_2 \times |\mathcal{U}|}$, which is pretrained on the entire UMLS KB using the knowledge embedding method TransE (Bordes et al., 2013). $\mathcal{U}$ denotes the entity vocabulary.

We merge the word embeddings with entity embeddings to feed them into a Multi-layer Perceptron (MLP):

$$\begin{aligned} \mathbf{h_i^q} &= \sigma(\mathbf{W_c w_i^q} + \mathbf{W_e e_i^q} + \mathbf{b}) \\ \mathbf{h_j^c} &= \sigma(\mathbf{W_c w_j^c} + \mathbf{W_e e_j^c} + \mathbf{b}) \end{aligned} \tag{1}$$

where σ is activation function, W_c, W_e, b are trainable parameters and h_i^q, h_j^c denote the integrated embeddings that contain information from both the word c_j and the entity e_j in the question and context respectively. For the word that is not mapped to an entity, e_j will be set to $\mathbf{0}$. The merged embeddings are used as the input to the base reader.

As shown in Figure 4, by adding a basic Knowledge Incorporation Module to the base model, we obtain around 5% increase of F1 score on the manipulated questions in the test set. This suggests that for questions that involve relations between medical concepts, external domain knowledge may be quite important.

Model	Existing Questions		Paraphrased Questions		New Questions		Overall		emrQA Relation	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
DocReader	58.33	71.62	38.09	57.28	29.41	35.35	40.00	53.27	86.94	94.85
ClinicalBERT	58.33	73.12	38.09	62.04	23.53	48.79	38.00	60.19	85.33	93.06

Table 5: Results of models when tested on new questions and unseen clinical notes (not in emrQA, but from MIMIC-III dataset). Performance drops around 40% compared with previously reported on the *Relation* test set, highlighting generalizability as an essential future direction for CliniRC.

5.2 Generalizing to unseen questions and documents

The aim of CliniRC is to build robust QA systems for doctors to retrieve information buried in clinical texts. When deploying a CliniRC system to a new environment (e.g., a new set of clinical records, a new hospital, etc.), it is infeasible to create new QA pairs for training every time. Thus, an ideal CliniRC system is able to generalize to unseen documents and questions after being fully trained.

To test the generalizability of models trained on emrQA (we focus on the *Relation* dataset here), our medical experts created 50 new questions that were not present in the emrQA dataset and extracted answers from unseen patient notes in the MIMIC-III (Johnson et al., 2016) dataset. This dataset consists of three types of questions: 12 questions were made from emrQA question templates but contain entities which do not appear in the training set (e.g., *"How was the diagnosis of acute cholecystitis made?"* was created from the template *"How was the diagnosis of |problem| made?"*). The other 38 questions have different forms from existing question templates: 21 paraphrase existing questions from emrQA (e.g., *"Was an edema found in the physical exam?")* was paraphrased from *"Does he have any evidence of |problem| in |test|?"*) and 17 are completely semantically different from the ones in the emrQA dataset (e.g., *"What chemotherapy drugs are being administered to the patient?"*).

As could be expected, we see in Table 5 that the more the new questions deviate from the original emrQA, the more the models struggle to answer them. We observe a performance drop of roughly 20% compared to the *Relation* test set on questions made from emrQA templates using MIMIC III clinical notes which were not in the original dataset. For question that are more significantly different, we notice an approximate 40% and 60% loss in F1 score when predicting paraphrased questions and entirely new questions respectively. This steep drop in performance for these new settings, especially for paraphrased and new questions, shows how much work there is to be done on this front and highlights generalizability as an important future direction in CliniRC. We also notice that ClinicalBERT works slightly better than the base model DocReader. The reason might be ClinicalBERT was pretrained on the MIMIC-III dataset, which might help the model have a better understanding of the context.

Summary. Based on these two aspects and our previous thorough analysis of the emrQA dataset, it is clear that better datasets are needed to advance CliniRC. Such datasets should be not only large-scale, but also less noisy, more diverse, and moreover allow researchers to systematically evaluate a model's ability to encode domain knowledge and to generalize to new questions and contexts.

6 Related Work

We present a brief overview of open-domain, biomedical and clinical question answering tasks, which are most related to our work:

Question Answering (QA) aims to automatically answer questions asked by humans based on external sources, such as Web (Sun et al., 2016), knowledge base (Yih et al., 2015; Sun et al., 2015) and free text (Chen et al., 2016). As an important type of QA, reading comprehension intends to answer a question after reading the passage (Hirschman et al., 1999). Recently, the release of large-scale RC datasets, such as CNN & Daily Mail (Hermann et al., 2015), Stanford Question-Answering Dataset (SQuAD) (Rajpurkar et al., 2016, 2018) makes it possible to solve RC tasks by building deep neural models (Hermann et al., 2015; Wang and Jiang, 2017; Seo et al., 2017; Chen et al., 2017).

More recently, contextualized word representations and pretrained language models, such as ELMo (Peters et al., 2018), GPT (Radford et al., 2018), BERT (Devlin et al., 2019), have been

demonstrated to be very useful in various NLP tasks including RC. By seeing diverse contexts in large corpora, these pretrained language models can capture the rich semantic meaning and produce more accurate and precise representations for words given different contexts. Even a simple classifier or score function built upon these pretrained contextualized word representations perform well in extracting answer spans (Devlin et al., 2019).

Biomedical and Clinical QA. Due to the lack of large-scale annotated biomedical or clinical data, QA and RC systems in these domains are often rule-based and heuristic feature-based (Lee et al., 2006; Niu et al., 2006; Athenikos and Han, 2010).

In recent years, BioASQ challenges (Tsatsaronis et al., 2012) proposed the Biomedical Semantic QA task, where the participants need to respond to each test question with relevant articles, snippets and exact answers. Šuster and Daelemans (2018) use summary points of clinical case reports to build a large-scale cloze-style dataset (CliCR), which is similar to the style of CNN & Daily Mail dataset. Jin et al. (2019b) presents PubMedQA, which extracts question-style titles and their corresponding abstracts as the questions and contexts respectively. A few QA pairs are annotated by human experts and most of them are annotated based a simple heuristic rule with "yes/no/maybe".

Due to the great power of contextualized word representations, pretrained language models also have been introduced to biomedical and clinical domain, e.g., BioELMo (Jin et al., 2019a), BioBERT (Lee et al., 2019), and ClinicalBERT (Alsentzer et al., 2019). They adopt similar architectures of the original models but pretrained on the medical and clinical corpus, such as PubMed articles and MIMIC-III (Johnson et al., 2016) clinical notes.

7 Conclusion

We study the Clinical Reading Comprehension (CliniRC) task with the recently created emrQA dataset. Our qualitative and quantitative analysis as well as exploration of the two desired aspects of CliniRC systems show that future clinical QA datasets should not only be large-scale but also less noisy and more diverse. Moreover, questions that involve complex relations and are across different domains should be included, and then more advanced external knowledge incorporation methods as well as domain adaptation methods can be carefully designed and systematically evaluated.

Acknowledgments

We thank our medical experts for their annotations. We thank Ping Zhang, Changchang Yin and anonymous reviewers for their helpful comments. This research was sponsored in part by the Patient-Centered Outcomes Research Institute Funding ME-2017C1-6413, the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, and Ohio Supercomputer Center (Center, 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S.Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Asma Ben Abacha and Dina Demner-Fushman. 2016. Recognizing question entailment for medical question answering. In *AMIA Annual Symposium Proceedings*, volume 2016, page 310.

Asma Ben Abacha and Dina Demner-Fushman. 2019. A question-entailment approach to question answering. *BMC bioinformatics*, 20(1):511.

Emily Alsentzer, John R Murphy, Willie Boag, Wei-Hung Weng, Di Jin, Tristan Naumann, and Matthew McDermott. 2019. Publicly available clinical bert embeddings. *NAACL Clinical NLP Workshop 2019*.

Sofia J Athenikos and Hyoil Han. 2010. Biomedical question answering: A survey. *Computer methods and programs in biomedicine*, 99(1):1–24.

Olivier Bodenreider. 2004. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 32(suppl_1):D267–D270.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NeurIPS'13*, pages 2787–2795.

Zied Bouraoui, Jose Camacho-Collados, and Steven Schockaert. 2020. Inducing relational knowledge from bert. *AAAI'20*.

Brian L Cairns, Rodney D Nielsen, James J Masanz, James H Martin, Martha S Palmer, Wayne H Ward, and Guergana K Savova. 2011. The mipacq clinical question answering system. In *AMIA annual symposium proceedings*, volume 2011, page 171.

YongGang Cao, Feifan Liu, Pippa Simpson, Lamont Antieau, Andrew Bennett, James J Cimino, John Ely,

and Hong Yu. 2011. Askhermes: An online question answering system for complex clinical questions. *Journal of biomedical informatics*, 44(2):277–288.

Ohio Supercomputer Center. 1987. Ohio supercomputer center.

Danqi Chen, Jason Bolton, and Christopher D Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. In *ACL'16*, pages 2358–2367.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *ACL'17*, pages 1870–1879.

Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. In *EMNLP'16*, pages 551–561.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL'19*, pages 2978–2988.

Dina Demner-Fushman, Wendy W Chapman, and Clement J McDonald. 2009. What can natural language processing do for clinical decision support? *Journal of biomedical informatics*, 42(5):760–772.

Joshua C Denny, Anderson Spickard III, Kevin B Johnson, Neeraja B Peterson, Josh F Peterson, and Randolph A Miller. 2009. Evaluation of a method to identify and categorize section headers in clinical documents. *Journal of the American Medical Informatics Association*, 16(6):806–815.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT'19*, pages 4171–4186.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A Smith. 2018. Annotation artifacts in natural language inference data. In *NAACL-HLT'18*, pages 107–112.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NeurIPS'15*, pages 1693–1701.

Lynette Hirschman, Marc Light, Eric Breck, and John D Burger. 1999. Deep read: A reading comprehension system. In *ACL'1999*, pages 325–332.

Qiao Jin, Bhuwan Dhingra, William Cohen, and Xinghua Lu. 2019a. Probing biomedical embeddings from language models. In *NAACL'19 RepEval2019 Workshop*, pages 82–89.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. 2019b. Pubmedqa: A dataset for biomedical research question answering. In *EMNLP-IJCNLP'19*, pages 2567–2577.

Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL'17*, pages 1601–1611.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*.

Minsuk Lee, James Cimino, Hai Ran Zhu, Carl Sable, Vijay Shanker, John Ely, and Hong Yu. 2006. Beyond information retrievalmedical question answering. In *AMIA annual symposium proceedings*, volume 2006, page 469.

Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. 2019. Scispacy: Fast and robust models for biomedical natural language processing. In *ACL BioNLP Workshop 2019*.

Yun Niu, Xiaodan Zhu, and Graeme Hirst. 2006. Using outcome polarity in sentence extraction for medical question-answering. In *AMIA Annual Symposium Proceedings*, volume 2006, page 599.

Anusri Pampari, Preethi Raghavan, Jennifer Liang, and Jian Peng. 2018. emrqa: A large corpus for question answering on electronic medical records. In *EMNLP'18*, pages 2357–2368.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL-HLT'18*, pages 2227–2237.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *EMNLP-IJCNLP'19*, pages 2463–2473.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *Technical report, OpenAI*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you dont know: Unanswerable questions for squad. In *ACL'18*, pages 784–789.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP'16*, pages 2383–2392.

S Trent Rosenbloom, Joshua C Denny, Hua Xu, Nancy Lorenzi, William W Stead, and Kevin B Johnson. 2011. Data from clinical notes: a perspective on the tension between structure and flexible documentation. *Journal of the American Medical Informatics Association*, 18(2):181–186.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hananneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. In *ICLR'17*.

Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *WWW'16*, pages 771–782.

Huan Sun, Hao Ma, Wen-tau Yih, Chen-Tse Tsai, Jingjing Liu, and Ming-Wei Chang. 2015. Open domain question answering via semantic enrichment. In *WWW'15*, pages 1045–1055.

Simon Šuster and Walter Daelemans. 2018. Clicr: A dataset of clinical case reports for machine reading comprehension. In *NAACL-HLT'18*, pages 1551–1563.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A machine comprehension dataset. In *ACL'17 RepL4NLP Workshop*, pages 191–200.

George Tsatsaronis, Michael Schroeder, Georgios Paliouras, Yannis Almirantis, Ion Androutsopoulos, Eric Gaussier, Patrick Gallinari, Thierry Artieres, Michael R Alvers, Matthias Zschunke, et al. 2012. Bioasq: A challenge on large-scale biomedical semantic indexing and question answering. In *2012 AAAI Fall Symposium Series*.

Shuohang Wang and Jing Jiang. 2017. Machine comprehension using match-lstm and answer pointer. *ICLR'17*.

Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. 2017. Gated self-matching networks for reading comprehension and question answering. In *ACL'17*, pages 189–198.

Yanshan Wang, Liwei Wang, Majid Rastegar-Mojarad, Sungrim Moon, Feichen Shen, Naveed Afzal, Sijia Liu, Yuqun Zeng, Saeed Mehrabi, Sunghwan Sohn, et al. 2018. Clinical information extraction applications: a literature review. *Journal of biomedical informatics*, 77:34–49.

Qiang Wei, Amy Franklin, Trevor Cohen, and Hua Xu. 2018. Clinical text annotation–what factors are associated with the cost of time? In *AMIA Annual Symposium Proceedings*, volume 2018, page 1552.

Jonathan D Wren. 2011. Question answering systems in biology and medicinethe time is now. *Bioinformatics*, 27(14):2025–2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP'18*, pages 2369–2380.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL'15*, pages 1321–1331.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. 2018. Qanet: Combining local convolution with global self-attention for reading comprehension. *ICLR'18*.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced language representation with informative entities. In *ACL'19*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV'15*, pages 19–27.

	Medication	Relation
# Train (Q / C)	154,684 /182	621,428 / 296
# Dev (Q / C)	23,081 / 26	101,700 / 42
# Test (Q / C)	45,192 / 53	181,464 / 85
Total	222,957 / 261	904,592 / 423

Table A1: Statistics of train, dev, test set of the *Medication* and *Relation* datasets.

A Experimental Set-up

We split the two datasets *Medication* and *Relation* based on the documents (clinical texts) into train, dev, test with the ratio 7:1:2. The statistics are shown in Table A1.

We adopt Exact Match (EM) and F1 score (F1) as our evaluation metrics, same as the open-domain RC (Rajpurkar et al., 2016). We use SQuAD v1.1 official evaluation script [1] to evaluate all the models. All the models used in the paper, BiDAF [2], DocReader [3], QANet [4], BERT [5], BioBERT [6], ClinicalBERT [7] are run based on the implementations listed here and strictly followed the instructions.

For reproducibility, we list all the key hyperparaters we use for each method in the Table A2.

We implement our Knowledge Incorporation Module based on DocReader implementations. Entity embeddings are pretrained using TransE (Bordes et al., 2013) with the dimension of 100. The hyperparameters are kept same as the DocReader. All the models are run on NVIDIA GeForce GTX 1080 GPUs. We save the best model (with the highest EM) on the dev set and use it for test set.

B Performance on Shorter Contexts

Using the entire clinical record as the context might be too long for models to capture sequential information. We also try to split the entire record into different sections (e.g., "medical history", "family history") based on some heuristic measures. Specifically, in order to split the clinical notes into sections, we notice that most sections begin with easily identifiable headers. To detect these headers we use a combination of heuristics such as whether the line contains colons, all uppercase formatting

[1] https://rajpurkar.github.io/SQuAD-explorer/
[2] https://github.com/allenai/bi-att-flow
[3] https://github.com/facebookresearch/DrQA
[4] https://github.com/BangLiu/QANet-PyTorch
[5] https://github.com/google-research/bert
[6] https://github.com/dmis-lab/biobert
[7] https://github.com/EmilyAlsentzer/clinicalBERT

Method	Hyper-parameters Setting
DocReader	epoch: 30; batch-size: 16; test-batch-size:16; droput-rate:0.4; doc-layers: 3; question-layers: 3; grad-clipping: 10; tune-partial: 1000; max-len: 30; the others are set as default
BiDAF	init_lr: 0.001; batch-size:6; num_epochs: 2; cluster: True; len-opt: True; word_count_th: 10; char_count_th: 50; sent_size_th: 4000; num_sents_th: 500; ques_size_th: 30; word_size_th: 30; para_size_th: 4000; the others are set as default;
QANet	batch-size: 4; lr: 0.001; grad-clip: 5; use-ema: True; epoch: 30; para_limit: 4000; ques_limit: 30; ans_limit: 30; char_limit: 40; num-head:1; the others are set as defalut;
BERT-base **BioBERT** **ClinicalBERT**	train_batch_size: 6; learning_rate: 3e-5; num_train_epochs: 3.0; max_seq_length: 384; doc_stride: 128; the others are set as default;

Table A2: Hyperparameters settings for all the methods used in the experiments.

Dataset	Model	Dev		Test	
		EM	F1	EM	F1
medication	DocReader	32.19	76.21	33.45	77.08
	ClinicalBERT	30.16	74.81	32.18	75.79
relation	DocReader	87.21	94.32	87.54	94.97
	ClinicalBERT	85.46	93.92	85.67	93.14

Table A3: Performance of the two models on the shorter context setting.

or phrases found in a list of clinical headers taken from SecTag (Denny et al., 2009). We then select the section that contains the answer as the context (∼100 words avg). We select DocReader and ClinicalBERT as representative methods and re-run them on the modified shorter context. The results are shown in Table A3. The performance of the two models is improved compared with the performance of models built on the whole record (long context). However, ClinicalBERT still does not outperform DocReader in this setting, indicating that longer context may not explain why BERT models do not win on this dataset or that shortening context in a such manner might break long dependencies.

This experiment setting may also inspire future research on "Open Clinical Reading Comprehension". Given that patients often have multiple clini-

Distribution of Easy/Hard Questions			Easy		Hard		Total	
		Medication	33,037 (73.1%)		12,155 (26.9%)		45,192 (100%)	
		Relation	165,271 (91.1%)		16,193 (8.9%)		181,464 (100%)	
Results		**Model**	Easy		Hard		Total	
			EM	F1	EM	F1	EM	F1
	Medication	DocReader (Chen et al., 2017)	30.25	73.78	13.26	61.46	25.68	70.45
		ClinicalBERT (Alsentzer et al., 2019)	28.25	72.02	12.64	60.98	24.06	69.05
	Relation	DocReader (Chen et al., 2017)	87.66	95.39	79.85	89.62	86.94	94.85
		ClinicalBERT (Alsentzer et al., 2019)	86.06	93.71	78.09	86.57	85.33	93.06

Table A4: Performance of DocReader and ClinicalBERT on the easy/hard questions split.

cal records, it may not be feasible to jointly use all of them as context for one question. Given multiple records for one patient (instead of just one) and a question, the model would first need to retrieve the most relevant paragraphs and do reading comprehension on each of them or find clever ways to merge them. Such a setting would be interesting for future CliniRC datasets to explore.

C Easy/Hard Questions Split

We partition the questions into Easy and Hard. Specifically, we first train a BiLSTM reader and do the prediction on the test set. We obtain the performance of each question template by averaging the performance of all the questions made by this template (such template and question mappings are included in the emrQA dataset). Question templates that obtain higher performance than the overall performance are labeled as "Easy" and "Hard" otherwise. Then we map the difficulty level of question templates back to each question. The reason why we focus on splitting on the question template level is that we can avoid some random noise (e.g., random errors produced by the model on some questions). Also, we release the difficulty level of each question template so that users can easily know which questions are easy or hard and do not need to run a base model to obtain such mappings again. Distributions of easy/hard questions and results of the two selected models are shown in Table A4.

Learning a Cost-Effective Annotation Policy for Question Answering

Bernhard Kratzwald[♢♠] **Stefan Feuerriegel**[♢] **Huan Sun**[♠]
[♢] Chair of Management Information Systems, ETH Zurich
[♠] Department of Computer Science and Engineering, The Ohio State University
{bkratzwald, sfeuerriegel}@ethz.ch sun.397@osu.edu

Abstract

State-of-the-art question answering (QA) relies upon large amounts of training data for which labeling is time consuming and thus expensive. For this reason, customizing QA systems is challenging. As a remedy, we propose a novel framework for annotating QA datasets that entails learning a cost-effective annotation policy and a semi-supervised annotation scheme. The latter reduces the human effort: it leverages the underlying QA system to suggest potential candidate annotations. Human annotators then simply provide binary feedback on these candidates. Our system is designed such that past annotations continuously improve the future performance and thus overall annotation cost. To the best of our knowledge, this is the first paper to address the problem of annotating questions with minimal annotation cost. We compare our framework against traditional manual annotations in an extensive set of experiments. We find that our approach can reduce up to 21.1% of the annotation cost.

1 Introduction

Question answering (QA) based on textual content has attracted a great deal of attention in recent years (Chen et al., 2017; Lee et al., 2018, 2019; Xie et al., 2020). In order for state-of-the-art QA models to succeed in real applications (e.g., customer service), there is often a need for large amounts of training data. However, manually annotating such data can be extremely costly. For example, in many realistic scenarios, there exists a list of questions from real users (e.g., search logs, FAQs, service-desk interactions). Yet, annotating such questions is highly expensive (Nguyen et al., 2016; He et al., 2018; Kwiatkowski et al., 2019): it requires the screening of a text corpus to find the relevant document(s) and subsequently screening the document(s) to identify the answering text span(s).

Motivated by the above scenarios, we study cost-effective annotation for question answering, whereby we aim to *accurately*[1] annotate a given set of user questions *with as little cost as possible*. Generally speaking, there has been extensive research on how to reduce effort in the process of data labeling (Haffari et al., 2009). For example, active learning for a variety of machine learning and NLP tasks (Siddhant and Lipton, 2018) aims to select a small, yet highly informative, subset of samples to be annotated. The selection of such samples is usually coupled with a particular model, and thus, the annotated samples may not necessarily help to improve a different model (Lowell et al., 2019). In contrast, we aim to annotate *all* given samples at low cost and in a manner that can subsequently be used to develop any advanced model. This is particularly relevant in the current era, where a dataset often outlives a particular model (Lowell et al., 2019). Moreover, there has also been some research into learning from distant supervision (Xie et al., 2020) or self-supervision (Sun et al., 2019). Despite being economical, such approaches often produce inaccurate or noisy annotations. In this work, we seek to reduce annotation costs without compromising the resulting dataset quality.

We propose a novel annotation framework which learns a cost-effective policy for choosing between different annotation schemes, namely the conventional manual annotation scheme (MAN) and a semi-supervised annotation scheme (SEM). Unlike the manual scheme, SEM does not require humans to screen a text corpus or document(s) in order to retrieve annotations. Instead, it leverages an initialized QA system, which can predict top-n candidate annotations for documents or answer spans and asks humans to provide binary feedback (e.g., cor-

[1] By "accurate," we mean that the resulting annotations will be of a similar quality to those from conventional manual annotation.

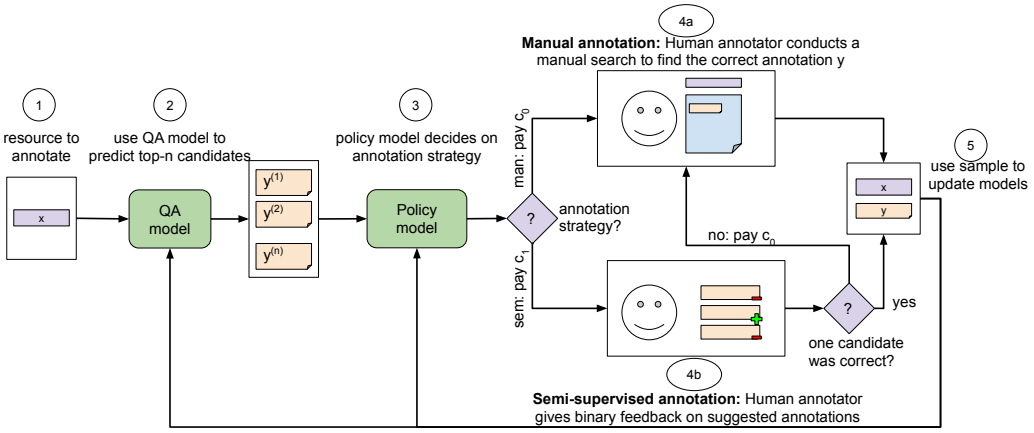


Figure 1: High-level overview of our framework: We leverage a QA model to predict candidate annotations for a given resource (e.g., x stands for a question or question-document pair, while y is the document or answer span). A policy model decides upon whether to invoke a MAN or SEM scheme based on those predictions. In the event that the semi-supervised strategy fails, we switch back to a manual annotation scheme. Finally, we use the annotated sample to update both the QA model and the policy model.

rect or incorrect) to the candidates. While this annotation scheme comes at a low cost, it fails when human annotators mark all candidates as incorrect. In such cases, the annotation cost has already been incurred and cannot be recouped. In order to produce an annotation, one must then draw upon the manual scheme (see Fig. 1), in which case the policy would have been more effective if it had chosen the manual annotation scheme instead. *Therefore, how to choose the best annotation scheme for each question is the challenge we must address for this task.*

To tackle the above challenge, we propose a novel approach for learning a cost-effective policy. Here the policy receives several candidates and decides on this basis which annotation scheme to invoke. We train the policy with a supervised objective and learn a cost-sensitive decision threshold. The inherent advantage of this method is that our policy immediately reacts to changing costs (without re-optimizing model parameters) and does not exceed the cost of conventional manual annotation. Our policy is updated iteratively as more annotations are obtained.

We compare our framework against conventional, manual annotations in an extensive set of experiments. We simulate the annotation of NaturalQuestions (Kwiatkowski et al., 2019), as it consists of real user questions from search logs. Models in our framework are initialized with an existing dataset (SQuAD, Rajpurkar et al., 2016) and, as more annotations on NaturalQuestions become available, the framework is continuously updated. We study the sensitivity of our framework to varying cost ratios between SEM and MAN. Our framework outperforms traditional manual annotation, even under conservative cost estimates for SEM, and in general reduces annotation costs in the range of 4.1% to 21.1%.

All source code is publicly available from github.com/bernhard2202/qa-annotation.

2 Related Work

Question answering: In this paper, we study cost-effective annotation for question answering over textual content. There have been extensive efforts to create large-scale datasets for text-based QA, which have facilitated the development of state-of-the-art neural network based models (e.g., Chen et al., 2017; Min et al., 2018; Lee et al., 2018; Kratzwald and Feuerriegel, 2018; Wang et al., 2018; Xie et al., 2020). Here we divide such datasets into two categories according to the way they were created: (1) Datasets whose questions were created by crowdsourcing during the annotation process. Prominent examples include the Stanford Question and Answer Dataset (SQuAD; Rajpurkar et al., 2016), HotPotQA (Yang et al., 2018), or NewsQA (Trischler et al., 2017). (2) "Natural" datasets in which real-world questions are a priori given. Here questions originate from, e.g., search logs or customer interactions. Prominent examples in this category include MS MARCO (Nguyen et al., 2016), DuReader (He et al., 2018), or Nat-

uralQuestions (Kwiatkowski et al., 2019). This paper focuses on the latter category, that is, annotating "natural" datasets in a more cost-effective fashion where a set of questions is given.

Active Learning: In the fields of machine learning and NLP, extensive research has been conducted on ways to reduce labeling effort (e.g., Zhu et al., 2008). For example, the objective of active learning is to select only a small subset that is highly informative (e.g., Haffari et al., 2009) for annotation. To this end, researchers have developed various techniques based on, e.g., model uncertainty (cf. Siddhant and Lipton, 2018), expected model change (Cai et al., 2013), or functions learned directly from data (e.g., Fang et al., 2017). However, the success of active learning is often coupled with a particular model and domain (Lowell et al., 2019). For instance, a dataset actively acquired with the help of an SVM model might underperform when used to develop an LSTM model. These problems become even more salient when complex black-box models are used in NLP tasks (cf. Chang et al., 2019). To summarize, active learning reduces annotation costs by deciding *which* samples should be annotated. In our approach, we aim to annotate *all* samples and study *how* we should annotate them in order to reduce costs. Thus, the two approaches are orthogonal and can be combined.

Learning from weak supervision and user feedback: Another approach to reducing annotation costs is changing full supervision to some form of weak (but potentially noisier) supervision. This has been adopted for various tasks such as machine translation (Saluja, 2012; Petrushkov et al., 2018; Clark et al., 2018; Kreutzer and Riezler, 2019), semantic parsing (Clarke et al., 2010; Liang et al., 2017; Talmor and Berant, 2018), or interactive systems that learn from user interactions (Iyer et al., 2017; Gur et al., 2018; Yao et al., 2019, 2020). For instance, Iyer et al. (2017) used users to flag incorrect SQL queries. In contrast, similar approaches for text-based question answering are scarce. Joshi et al. (2017) used noisy distant supervision to annotate the answer span and document for given trivia questions and their answers. Kratzwald and Feuerriegel (2019) designed a QA system that continuously learns from noisy user feedback after deployment. In contrast to these works, this paper studies the problem of reducing labeling cost while maintaining accurate annotations.

Quality estimation and answer triggering: In a broader sense, this work is related to the literature on translation quality estimation (e.g., Martins et al., 2017; Specia et al., 2013). The goal in such works is to estimate (and possibly improve) the quality of translated text. Similarly, in question answering researchers use means of quality estimation for answer triggering (Zhao et al., 2017; Kamath et al., 2020). Here, QA systems are given the additional option to abstain from answering a question when the best prediction is believed to be wrong. In our work, we estimate the quality of a set of suggested label candidates and, on the basis of these estimates we decide which annotation scheme to invoke.

3 Proposed Annotation Framework

We study the problem of reducing the overall cost for annotating every given question $[q_1, \ldots, q_m]$. Specifically, our objective is to obtain the corresponding question-document-answer triples $\langle q_i, d_i, s_i \rangle$. In this paper, the natural language question q_i is given, while we want to obtain the following annotations: the document from a text corpus $d_i \in \mathcal{D}$ that contains the answer and the correct answer span s_i within the document d_i.

3.1 Framework Overview

Fig. 1 provides an overview of our framework for a cost-effective annotation of QA datasets. The framework comprises two main components: a **QA model** is used to suggest candidates for a resource to annotate while a **policy model** decides which annotation scheme to invoke (i.e., action). Our framework makes use of two annotation schemes: a traditional **manual annotation scheme (MAN)** and our **semi-supervised annotation scheme (SEM)**. Both annotation schemes incur different costs and, hence, the learning task is to find and update a cost-effective policy π for making that decision.

QA model: We define Ω as an arbitrary QA model over a text corpus $\mathcal{D}$ with the following properties. First, the model can be trained from annotated data samples, e.g., $\Omega \leftarrow \text{train}\left(\{\langle q_i, d_i, s_i \rangle\}_{0<i<\ldots}\right)$. Second, for a given question the model can predict a number of top-n documents likely to contain the answer, i.e., $\Omega^D : q \rightarrow [d^{(1)}, \ldots, d^{(n)}] \in \mathcal{D}$. Third, for a given question-document pair the model can predict a number of top-n answer spans, i.e., $\Omega^S : \langle q, d \rangle \rightarrow [s^{(1)}, \ldots, s^{(n)}]$. These properties

are fulfilled by recent QA systems dealing with textual content (e.g., Chen et al., 2017; Wang et al., 2018).

Policy model: For every question, we distinguish two policy models: a policy model π^D responsible for annotating documents and π^S for answer spans. For brevity, we sometimes drop the superscripts S and D and simply refer to them as π. The policy models decide whether a manual annotation scheme or rather our proposed semi-supervised annotation scheme is used, each of which is associated to different costs.

3.2 Annotation Schemes

Manual annotation (MAN) scheme: This scheme represents the status quo in which all annotations are determined manually. In order to annotate a question q_i, a human annotator must first manually search through the text corpus $\mathcal{D}$ in order to identify the document d_i that answers the question. In a second step, a human annotator manually reads through the document d_i and marks the answer span s_i.

We assume separate costs, which are fixed over time, for every annotation-level. The price of annotating a document for a given question is defined as c_0^D and the price of annotating an answer span to a given question-document tuple as c_0^S. We explicitly distinguish these costs as the tasks can be of differing difficulty.

Semi-supervised annotation (SEM) scheme: This scheme is supposed to reduce human effort by presenting candidates for annotation, so that only simple binary feedback is needed. In particular, human annotators no longer need to search through the entire document or corpus. Instead, we use the QA model Ω to generate a set of candidates (e.g., top-ranked documents or answer spans) and ask human annotators to give binary feedback in response (e.g., accept the candidate or reject it). This replaces the complex search task with a simpler form of interaction. As an example, to annotate the answer span for a question-document pair, the human annotator would not be required to read the entire document d_i, but only to determine which of the top-n answers provided by $\Omega^S(\langle q_i, d_i \rangle)$ are correct. We assume SEM costs c_1^D to annotate a document and c_1^S to annotate an answer span.

The SEM scheme should make annotations more straightforward, as providing binary feedback requires less time than reading through the texts. Hence, we assume that $c_1^S < c_0^S$ and $c_1^D < c_0^D$ hold. However, semi-supervised annotations might fail when none of the candidates is correct (i.e., the human annotators reject all candidates). In this case, our framework must revert to the MAN procedure in order to obtain a valid annotation. As a consequence, the associated cost will increase to the accumulated cost for both the SEM and the MAN schemes.

Note that, no matter which scheme is chosen in practice, all annotations are confirmed by human annotators and our resulting dataset will be equal in quality to those resulting from traditional annotation.

3.3 Annotation Costs

Both annotation schemes, MAN and SEM, incur different costs that further vary depending on whether annotation is provided at document level (c^D) or at answer span level (c^S). For annotating documents, the cost amounts to

$$c^D(a|q_i, d_i^*) = \begin{cases} c_a^D, \text{ if } a = 0 \text{ or } (a = 1 \text{ and} \\ \qquad\qquad d_i^* \in \Omega^D(q_i)), \\ c_0^D + c_1^D, \quad \text{otherwise} \end{cases} \tag{1}$$

where $a = \{0, 1\}$ is the selected annotation scheme and d_i^* is the ground-truth document annotation. Hence, $d_i^* \in \Omega^D(q_i)$ indicates the candidate set contains the ground-truth annotation and SEM is successful.
For annotating answer spans, the cost is given by

$$c^S(a|\langle q_i, d_i \rangle, s_i^*) = \begin{cases} c_a^S, \text{ if } a = 1 \text{ or } (a = 0 \text{ and} \\ \qquad\qquad s_i^* \in \Omega^S(\langle q_i, d_i \rangle)), \\ c_0^S + c_1^S, \quad \text{otherwise.} \end{cases} \tag{2}$$

Alternatively, we can write the cost function as a matrix of annotation costs (Tbl. 1). The diagonal entries reflect the costs paid for choosing the optimal scheme. The off-diagonals refer to the costs paid for a sub-optimal method (misclassification costs).

4 Learning a Cost-Effective Policy

4.1 Objective

We aim to minimize overall annotation cost via

$$\sum_i \mathbb{E}_{\pi^D, \pi^S} \left[c^D(a|q_i, d_i^*) + c^S(a|\langle q_i, d_i \rangle, s_i^*) \right]. \tag{3}$$

Annotation Costs for a Document d		
	Cost-optimal: MAN	Cost-optimal: SEM
Selected: MAN	c_0^D	c_0^D
Selected: SEM	$c_0^D + c_1^D$	c_1^D

Annotation Costs for an Answer Span s		
	Cost-optimal: MAN	Cost-optimal: SEM
Selected: MAN	c_0^S	c_0^S
Selected: SEM	$c_0^S + c_1^S$	c_1^S

Table 1: Costs for annotating documents (top) and answer spans (bottom). The costs depend on the selected annotation scheme (rows) and the scheme that would have been cost-optimal (columns).

It is important to see that the QA model and the policy model are intertwined, with both having an impact on Eq. 3. Updating the policy models learns the trade-off between SEM and MAN annotations and, hence, directly minimizes the overall costs. Updating the QA model Ω increases the number of times suggested candidates are correct and, therefore, the fraction of successful SEM annotations. For instance, when adapting to a new domain, only a small fraction of suggested candidate annotations are correct, limiting the effectiveness of the SEM annotation. However, as we annotate more samples, we improve Ω and thus more suggested candidate annotations will be correct. For this, we later specify suitable updates for both the QA model and the policy model.

4.2 Annotation Procedure and Learning

Our framework proceeds according to these nine steps when annotating a question q_i (see Alg. 1): First, we predict a number of top-n documents that would be shown to annotators in the case of SEM annotation (line 2). Next, we decide upon the annotation scheme conditional on the prediction from the QA model (line 3) and, based on the selected scheme, request the ground-truth document annotation (line 4). After receiving the ground-truth document and observing the annotation costs, we update our policy network in line 5 (see Sec. 4.3). Next, we predict a number of top-n answer span candidates for the question-document pair (line 6) and then decide upon the annotation scheme in line 7. After receiving the answer span annotation and observing a cost (line 8), we again update our policy model (line 9). Finally, we update the QA model with the newly annotated training sample in line 10 (see Sec. 4.4). In practice, both policy updates and QA model updates (lines 5, 9, and 10) are invoked after a batch of questions is annotated. Furthermore, we initialize all models with an existing dataset (e.g., SQuAD).

Algorithm 1: High-Level Procedure of Annotation and Learning

Input : list of questions $[q_1, \ldots, q_m]$ text corpus $\mathcal{D}$; QA model Ω; policy models π^D and π^S
Result: annotated dataset $\{\langle q_i, d_i, s_i\rangle\}_{0<i<m}$

1 **while** $i \leq m$ **do**
2 $\quad[d^{(1)}, \ldots, d^{(n)}] \leftarrow \Omega^D(q_i)$; predict top-$n$ documents
3 $\quad a \leftarrow \pi^D(q_i, [d^{(1)}, \ldots, d^{(n)}])$; decide upon annotation scheme
4 $\quad d_i \leftarrow annotate(q_i|a)$; annotate document
5 $\quad$ update π^D w.r.t. the observed costs $c^D(a|q_i, d_i)$;
6 $\quad[s^{(1)}, \ldots, s^{(n)}] \leftarrow \Omega^S(\langle q_i, d_i\rangle)$; predict top-$n$ answer candidates
7 $\quad a \leftarrow \pi^S(q_i, [s^{(1)}, \ldots, s^{(n)}])$; decide upon annotation scheme
8 $\quad s_i \leftarrow annotate(\langle q_i, d_i\rangle|a)$; annotate answer span
9 $\quad$ update π^S w.r.t. the observed costs $c^S(a|\langle q_i, d_i\rangle, s_i)$;
10 $\quad$ update QA model Ω with $\langle q_i, d_i, s_i\rangle$
11 **end**

4.3 Policy Updates

Updating our policy model proceeds in three steps. (1) We calculate whether the chosen action for past annotations was cost-optimal (i.e., whether the policy should have chosen the other scheme for annotation or not). (2) We use this information to update the policy model with a supervised binary classification objective. This trains the policy to predict the probability of an annotation scheme given a new sample $p(a|x)$ without taking costs into account. (3) We find a cost-sensitive decision threshold that chooses the optimal action with respect to the costs. All three steps are repeated after a full batch of samples has been annotated.

Separating the policy update and the cost-sensitive decision threshold has several benefits. First, we know from cost-sensitive classification that we can calculate an optimal threshold point for ground-truth probabilities $p(a|x)$ (c.f. Elkan, 2001; Ting, 2000). Therefore, we can focus our effort on determining probabilities as accurate as possible. Second, the decision threshold is calculated only from the costs c_a^S and c_a^D and, hence, if costs change, we do not need to re-estimate parameters but can directly adjust our policy.

(1) Finding the cost-optimal action: In order to train the policy with a supervised update, we require labels for the cost-optimal annotation scheme

for a given sample. If we choose the SEM annotation scheme, we immediately know whether the action was cost-optimal or not. This is due to the fact that, if the semi-supervised annotation fails, we have to switch to the MAN scheme to receive an annotation and pay both costs. On the other hand, if we choose the MAN scheme, we can observe the optimal action only after receiving the ground-truth annotation: We can then simply run the QA model and validate whether the annotation was contained within the top-n candidates. If so, the SEM action would have been the better choice; otherwise, choosing MAN would have been cost-optimal.

(2) Supervised model updates: Our policy model is a neural network with parameters θ that predicts an annotation scheme for a given sample x, i.e.,

$$p(a|x) = \text{NN}_\theta(x). \tag{4}$$

Note that we dropped the S and D indices here as both policies differ only in the neural network architecture used. We can then simply train the policy with a supervised binary cross-entropy loss given the cost-optimal action that we calculated beforehand. Since the SEM scheme is often sub-optimal in the beginning, the training data is highly imbalanced. Therefore, we down-sample past annotations with a sampling ratio of α such that our training data is equally balanced.

(3) Cost-sensitive decision threshold: Choosing the annotation scheme with the highest probability does not take the actual costs into account. For instance, if SEM annotations are much cheaper than MAN annotations, we want to choose the semi-supervised scheme even if its probability for success is low. More formally, we want to choose the annotation scheme a that has the lowest expected cost $R(a|x)$, i.e.,

$$R(a \mid x) = \sum_{a'} p(a' \mid x)\, c(a, a') \tag{5}$$

where $c(a, a')$ is the annotation cost for choosing scheme a when the optimal scheme was a' (see Tab. 1). Since we used down-sampling in our training, we have to calibrate the probabilities $p(a|x)$ with the sampling ratio α (Pozzolo et al., 2015). Assuming the calibrated probabilities are accurate, there exists an optimal classification threshold β (Elkan, 2001) that minimizes Eq. 5 and Eq. 3. Therefore, we define our policy as follows:

$$\pi(a|x, \alpha, \beta) = \begin{cases} 1, \text{ if } \frac{\alpha\, p(a=1|x)}{(\alpha-1)\, p(a=1|x)+1} \geq \beta \\ 0, \text{ otherwise.} \end{cases} \tag{6}$$

The optimal β can be derived from the classification cost matrix (Elkan, 2001) via

$$\beta = \frac{c(1,0) - c(0,0)}{c(1,0) - c(0,0) + c(0,1) - c(1,1)}, \tag{7}$$

again omitting superscripts D and S for brevity. Eq. 7 is then simplified as the fraction of SEM annotation costs to MAN annotation costs. Therefore, we can derive β at document level

$$\beta^D = c_1^D / c_0^D, \tag{8}$$

and answer span level

$$\beta^S = c_1^S / c_0^S. \tag{9}$$

4.4 Model Updates

Periodically updating the QA model Ω allows the framework to adapt to the question style and domain at hand during the annotation process. Therefore, we improve the top-n accuracy and the success rate of the SEM scheme over time. For practical reasons, we refrain from updating Ω after every annotation but periodically retrain the model after a batch of samples is annotated. In order to update the QA model, we can use the fully annotated QA samples in combination with a supervised objective.

5 Experimental Setup

In this section, we introduce our experimental setup and implementation details.

5.1 Datasets

We base our experiments on the NaturalQuestion dataset (Kwiatkowski et al., 2019). We choose this dataset as it is composed of about 300,000 real user questions posed to the Google search engine along with human-annotated documents and answer spans. Simulating the annotation of this dataset is similar to what would happen for domain customization of QA models in real practice (e.g., for search logs, FAQs, logs of past customer interactions). We focus on questions from the training-split that possess an answer span annotation and leave the handling of questions that do not have an answer for future work. The corpus for annotations

is fixed to the English Wikipedia,[2] containing more than 5 million text documents.

Simulation of annotations: Annotations in our experiments are simulated from the original dataset. If the framework chooses MAN annotation, we simply use the original annotation from the dataset. If a SEM annotation is chosen, we simulate users that give positive feedback only to the ground-truth document and to the answer spans where the text matches[3] the ground-truth annotation. We then construct the new annotation using the candidate with positive feedback. Since we simulate annotations, we conduct extensive experiments on how annotation costs influence the performance of our framework.

5.2 Baselines

To the best of our knowledge, there is no comparable prior work. Owing to this fact, we evaluate our framework against several customized baselines. First, we compare our approach against a manual annotation baseline in which we always invoke the full MAN method to annotate samples. This represents the traditional method of annotating QA datasets and thus our prime baseline. Second, we draw upon a clairvoyant oracle policy that always knows the optimal annotation method. We use this baseline to report an upper bound of the savings that our framework could theoretically achieve. Third, we use our framework without updates on the QA model Ω. This quantifies the cost-savings achieved by the interactive domain customization during annotation. Finally, we present a randomized baseline where the annotation scheme is decided by a randomized coin-toss.

5.3 Implementation Details

The QA model Ω is built as follows. We use a state-of-the-art BERT-based (Devlin et al., 2018) implementation of RankQA (Kratzwald et al., 2019). This combines a simple tf-idf-based information retrieval module with BERT as a module for machine comprehension. Both policy models π^D and π^S are implemented as three-layer feed-forward networks with dropout, ReLu activation, and a single output unit with sigmoid activation in the last layer. For the policy π^D, we use the information retrieval scores as input. For π^S, we use the statistical features of answer-span candidates as calculated by RankQA (Kratzwald et al., 2019) as input. We also experimented with convolutional neural networks directly on top of the last layer of BERT, but without yielding improvements that justified the additional model complexity. We initialize all models with the SQuAD dataset (see our supplements).

Hyperparameter setting: We set the number of candidates that are shown to annotators during a SEM annotation to $n = 5$. The policy networks decide upon the annotation method based on features of the $2n$ highest-ranked candidates, i.e., the top-10. The batch size for updates in Alg. 1 is set to 1,000 annotated questions. Details on hyperparameters of our QA and policy models are provided in the supplements.

6 Experimental Results

We group our experiments into three parts. First, we focus only on annotating the answer span for given question-document pairs, as this is the more challenging task.[4] Second, we carry out a sensitivity analysis in order to demonstrate how our framework adapts to different costs of SEM annotations and to show that we never exceed the cost of traditional annotation. Third, we evaluate our framework based on the annotation of a full dataset, including both answer span and document annotations, in order to quantify savings in practice by using our framework.

6.1 Performance on Answer Span Annotations

The annotation framework was used to annotate 45 batches of question-document pairs with the corresponding answer spans. The annotation costs are set to one price-unit for each MAN annotation and one third of the unit for each SEM annotation. (In the next section, we carry out an extensive sensitivity analysis where the ratio for annotation costs between MAN and SEM is varied.)

In Fig. 2 (left), we plot the average annotation costs in every batch with a dashed line, together with a running mean depicted as a solid line. Compared to conventional, manual annotation, our framework successfully reduces annotation cost by around 15% after only 20 batches. We further

[2] We extracted articles from the Wikipedia dump collected in October 2019, as this is close to the time period in which the NaturalQuestions dataset was constructed.

[3] Here we only count exact matches.

[4] Manually finding an answer span involves reading a document in depth. Manual document annotation is easier, as it can be supported with tools such as search engines. In such a case, our framework could still be used for answer span annotation, as is shown in Section 6.1.

compare it with an oracle policy that always picks the best annotation method. The latter provides a hypothetical upper bound according to which approximately 40–45% of annotation cost could be saved. Finally, we show the performance of our framework without updates of the QA model Ω. Here we can see that its improvement over time is lower, as the framework is not capable of adapting to the question style and domain used during annotation. In sum, our framework is highly effective in reducing annotation cost.

Fig. 2 (right) shows how many samples we could annotate (y-axis) with a restricted budget (x-axis). For instance, assume we have a budget of 40k price units available for annotation. Conventional, manual annotation would result in exactly 40k annotated samples as we fixed the cost for each MAN annotation to one unit. With the same budget, our annotation framework with semi-supervised annotations succeeds in annotating an additional ∼9,000 samples.

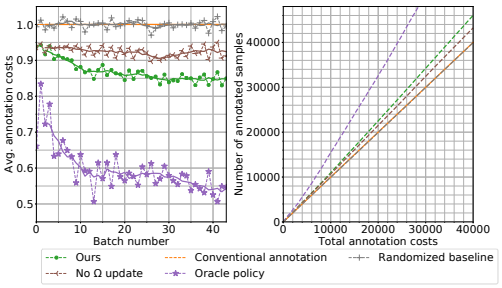


Figure 2: Left: average annotation costs in every batch as a dashed line, together with a running mean as a solid line. Right: how many samples we could annotate (y-axis) with a restricted available budget (x-axis).

6.2 Cost-Performance Sensitivity Analysis

The advantage of our framework over manual annotations depends on the cost ratio between the SEM and MAN schemes. In order to determine this, we identify the cost-range in which our framework is profitable as a function of SEM annotation costs. We study this via the following experiment: we repeatedly annotate 40k samples and keep the MAN annotation costs fixed to one price unit, while we increase the costs of smart annotations from 0.05 to 0.95 in increments of 0.05. Finally, we measure the average annotation costs for a single sample; see Fig. 3 (left).

Fig. 3 (left) demonstrates that our framework effectively lowers annotation costs when the price for SEM annotations drops below 0.6 as compared to manual annotations, which are fixed to one price-unit. Most notably, even when SEM annotations become expensive and almost equal the costs of MAN annotations, the average annotation costs do not exceed those of strictly manual annotation. This can be attributed to our cost-sensitive decision threshold, which does not require exploration as in reinforcement learning, but directly sets the threshold in Eq. 6 sufficiently high.

In Fig. 3 (right), we again show the number of samples that were annotated with a restricted budget of 40k price units. We marked the absolute gain in number of samples over traditional annotation in the plot. The benefit of our framework becomes evident once again when the ratio of SEM annotation costs to MAN annotations costs falls below 0.6.

To summarize, our framework is highly cost-effective: it reduces overall annotation costs or, alternatively, increases the number of annotated samples under a restricted budget if annotation costs of SEM are approximately half those of MAN. If the costs are less than half those of MAN annotation, the benefits are especially pronounced. Even if this assumption does not hold, our framework never exceeds the costs of manual annotation and never results in fewer annotated samples.

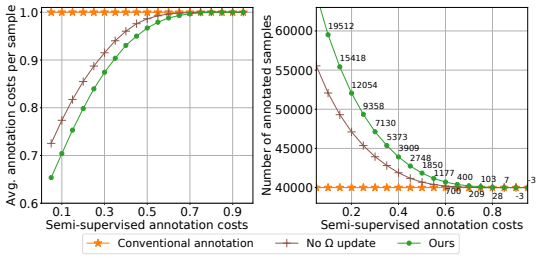


Figure 3: Left: average annotation costs when varying the ratio of SEM over MAN annotation costs. Right: number of samples that were annotated with a restricted budget for a given SEM annotation cost.

6.3 Performance on Full Dataset Annotation

In the last experiment, we simulate a complete annotation of the NaturalQuestions dataset, including annotations at both document level and answer span level. By annotating a complete dataset, we want to quantify the savings of our framework in practice. We again set the cost of each MAN annotation to one price-unit and repeated the experiment three times by setting the SEM annotation cost (c_1)

	Document-level			Answer span-level			Overall		
	$c_1 = 1/4$	$c_1 = 1/3$	$c_1 = 1/2$	$c_1 = 1/4$	$c_1 = 1/3$	$c_1 = 1/2$	$c_1 = 1/4$	$c_1 = 1/3$	$c_1 = 1/2$
Traditional Annotation	102.4	102.4	102.4	102.4	102.4	102.4	204.8	204.8	204.8
Ours	79.8	85.6	98.0	81.8	87.0	98.3	161.6	172.7	196.3
	(22.1%)	**(14.9%)**	**(4.2%)**	**(20.1%)**	**(14.9%)**	**(4.0%)**	**(21.1%)**	**(15.7%)**	**(4.1%)**

Table 2: Overall cost ($\times 10^3$ price unit) for annotating the NaturalQuestions dataset using our framework vs. conventional manual annotation for different SEM costs (c_1). Improvements are shown in parenthesis.

to one quarter, one third, and one half of the price unit. The results are shown in Tbl. 2. Depending on relative cost ratio c_1, we are able to save between 4.1% and 21.1% percent of the overall annotation cost. This amounts to a total of 40,000 to 8,000 price units.[5]

7 Discussion and Future Work

We assume for the purposes of this study that questions have an answer span contained in a single document and leave an extension to multi-hop questions and unanswerable questions to future research. The robustness of our framework is demonstrated in an extensive set of simulations and experiments. We deliberately choose to leave experiments including real human annotators to future research for the following reason. Outcomes of such an experiment would be sensitive to the design of the user interface as well as the study design itself. In this paper, we want to put the emphasis on the methodological innovation of our framework and the novel annotation scheme itself.

On the other hand, experiments involving real users would provide valuable insights concerning the annotation costs and the quality of a dataset annotated with our method. Furthermore, it would be worth investigating how inter-annotator agreement or potential human biases manifest in traditional datasets as compared to those generated with our framework.

8 Conclusion

We presented a novel annotation framework for question answering based on textual content, which learns a cost-effective policy to combine a manual annotation scheme with a semi-supervised annotation scheme. Our framework annotates all given questions accurately while limiting costs as much as possible. We show that our framework never incurs higher costs than traditional manual annotation. On the contrary, it achieves substantial savings. For example, it reduces the overall costs by about 4.1% when SEM annotations cost about half of MAN annotations. When that ratio is lowered to one fourth, our framework can reduce the total costs by up top 21.1%. We think that our framework could contribute to more accessible annotation of datasets in the future and possibly even be extended to other fields and applications in natural language processing.

[5] In these experiments, we obtain the overall cost by directly adding the costs of the two levels. Note that the cost can be different for document and answer span annotations, and that in such cases, our framework can still save costs at each level as shown in the table, although we cannot directly add up the costs as an overall sum.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. We are also deeply grateful to Ziyu Yao for her valuable feedback and help in brainstorming and formalizing this idea. This research was partly supported by the Army Research Office under cooperative agreements W911NF-17-1-0412, and by NSF Grant IIS1815674 and NSF CAREER #1942980. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

W. Cai, Y. Zhang, and J. Zhou. 2013. Maximizing expected model change for active learning in regression. In *IEEE International Conference on Data Mining*, pages 51–60.

Haw-Shiuan Chang, Shankar Vembu, Sunil Mohan, Rheeya Uppaal, and Andrew McCallum. 2019. Overcoming practical issues of deep active learning and its applications on named entity recognition. *arXiv preprint arXiv:1911.07335*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1914–1925.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Conference on Computational Natural Language Learning (CoNLL)*, pages 18–27.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NACL)*.

Charles Elkan. 2001. The foundations of cost-sensitive learning. In *International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, page 973–978.

Meng Fang, Yuan Li, and Trevor Cohn. 2017. Learning how to active learn: A deep reinforcement learning approach. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 595–605.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue based structured query generation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1339–1349.

Gholamreza Haffari, Maxim Roy, and Anoop Sarkar. 2009. Active learning for statistical phrase-based machine translation. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NACL)*, pages 415–423.

Wei He, Kai Liu, Jing Liu, Yajuan Lyu, Shiqi Zhao, Xinyan Xiao, Yuan Liu, Yizhong Wang, Hua Wu, Qiaoqiao She, Xuan Liu, Tian Wu, and Haifeng Wang. 2018. DuReader: a Chinese machine reading comprehension dataset from real-world applications. In *Workshop on Machine Reading for Question Answering*, pages 37–46.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 963–973.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1601–1611.

Amita Kamath, Robin Jia, and Percy Liang. 2020. Selective question answering under domain shift. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5684–5696.

Bernhard Kratzwald, Anna Eigenmann, and Stefan Feuerriegel. 2019. Rankqa: Neural question answering with answer re-ranking. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Bernhard Kratzwald and Stefan Feuerriegel. 2018. Adaptive document retrieval for deep question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 576–581.

Bernhard Kratzwald and Stefan Feuerriegel. 2019. Learning from On-Line User Feedback in Neural Question Answering on the Web. In *The World Wide Web Conference (WWW)*, pages 906–916.

Julia Kreutzer and Stefan Riezler. 2019. Self-regulated interactive sequence-to-sequence learning. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 303–315.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics (TACL)*.

Jinhyuk Lee, Seongjun Yun, Hyunjae Kim, Miyoung Ko, and Jaewoo Kang. 2018. Ranking Paragraphs for Improving Answer Recall in Open-Domain Question Answering. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 565–569.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6086–6096.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 23–33.

David Lowell, Zachary C. Lipton, and Byron C. Wallace. 2019. Practical obstacles to deploying active learning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 21–30.

André F. T. Martins, Marcin Junczys-Dowmunt, Fabio N. Kepler, Ramón Astudillo, Chris Hokamp, and Roman Grundkiewicz. 2017. Pushing the limits of translation quality estimation. *Transactions of the*

Association for Computational Linguistics (TACL), 5:205–218.

Sewon Min, Victor Zhong, Richard Socher, and Caiming Xiong. 2018. Efficient and Robust Question Answering from Minimal Context over Documents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1725–1735.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Pavel Petrushkov, Shahram Khadivi, and Evgeny Matusov. 2018. Learning from chunk-based feedback in neural machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 326–331.

A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. 2015. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Empirical Methods in Natural Language Processing (EMNLP) Language Processing*, pages 2383–2392.

Avneesh Saluja. 2012. Machine Translation with Binary Feedback : a Large-Margin Approach. In *Biennial Conference of the Association for Machine Translation in the Americas*.

Aditya Siddhant and Zachary C. Lipton. 2018. Deep Bayesian Active Learning for Natural Language Processing: Results of a Large-Scale Empirical Study. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2904–2909.

Lucia Specia, Kashif Shah, Jose G.C. de Souza, and Trevor Cohn. 2013. QuEst - a translation quality estimation framework. In *Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 79–84.

Yibo Sun, Duyu Tang, Nan Duan, Yeyun Gong, Xiaocheng Feng, Bing Qin, and Daxin Jiang. 2019. Neural semantic parsing in low-resource settings with back-translation and meta-learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Conference of the North American Chapter of the Association for Computational Linguistic (NACL)*, pages 641–651.

Kai Ming Ting. 2000. A comparative study of cost-sensitive boosting algorithms. In *International Conference on Machine Learning (ICML)*, pages 983–990.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A machine comprehension dataset. In *Workshop on Representation Learning for NLP*, pages 191–200.

Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerald Tesauro, Bowen Zhou, and Jing Jiang. 2018. R^3: Reinforced Reader-Ranker for Open-Domain Question Answering. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Yuqing Xie, Wei Yang, Luchen Tan, Kun Xiong, Nicholas Jing Yuan, Baoxing Huai, Ming Li, and Jimmy Lin. 2020. Distant supervision for multi-stage fine-tuning in retrieval-based question answering. In *The Web Conference (WWW)*, page 2934–2940.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 2369–2380.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5447–5458.

Ziyu Yao, Yiqi Tang, Wen-tau Yih, Huan Sun, and Yu Su. 2020. An imitation game for learning semantic parsers from user interaction. *arXiv preprint arXiv:2005.00689*.

Jie Zhao, Yu Su, Ziyu Guan, and Huan Sun. 2017. An end-to-end deep framework for answer triggering with a novel group-level objective. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1276–1282.

Jingbo Zhu, Huizhen Wang, Tianshun Yao, and Benjamin K Tsou. 2008. Active learning with sampling by uncertainty and density for word sense disambiguation and text classification. In *International Conference on Computational Linguistics (Coling)*, pages 1137–1144.

Appendix

A Source Code

All source code is available from github.com/bernhard2202/qa-annotation.

B Details on the QA Model

We use the same hyperparemter configuration as reported in Kratzwald et al. (2019) without further fine-tuning. The model was initialized by training on the training split of the SQuADv1.1. dataset (Rajpurkar et al., 2016).

Parameter	Values
Dropout z	0.0, **0.3**, 0.5
Hidden units k	32 64 **128**
Learning rate	**0.0001**, 0.0005, 0.001
Epochs	15, 20, **25**

Table 3: Values used for gridsearch in hyperparameter tuning for the policy π^S

Parameter	Values
Dropout z	0.0, **0.3**, 0.5
Hidden units k	32 64 **128**
Learning rate	**0.0001**, 0.0005, 0.001
Epochs	15, 20, **25**

Table 4: Values used for gridsearch in hyperparameter tuning for the policy π^D

C Details on the Policy Model

The policy models π^D and π^S are implemented as feed forward networks composed of a dense layer with k output units and relu actvation, a dropout layer with dropout probability z, a second dense layer with $k/2$ outputs and relu activation, a dropout layer with dropout probability z, and a dense layer with a single output and sigmoid activation.

Initialization: for the first batch of annotations we initialize the policy models on SQuAD. After the first batch is annotated we only use the new data for policy updates.

Hyperparameter search: We tune hyperparamters on the SQuAD dataset using gridsearch with the values displayed in Tab. 3 and Tab. 4. Bold values mark final choices. We annotated the first 10 batches of SQuAD and choose the hyperparamters that had the lowest anntation cost. No hyperparemter tuning or architecture search was performed on the NaturalQuestions dataset which our experiments are based on.

D Estimation of Real Annotation Costs

In order to provide additional insights on the actual annotation costs involving real users we conducted a pre-test on Amazon MTURK. For this we showed a textual explanation of the MAN and SEM annotation scheme to workers and provided them with mockups for both inputs (answer-span annotation). Next, we asked 40 workers to report how much money they think would be a fair compensation for each of the tasks on a scale of one to ten. Workers reported on average a compensation of \$5.9 for MAN annotations and \$3.2 for SEM annotations. This ratio falls into the range where we make profits using our framework.

E System

All experiments were conducted with a Nvidia Titan Xp GPU on a Server with 192GB DDR4 RAM and two 10 Core Intel Xeon Silver 4210 2.2GHz Processors.

Model-based Interactive Semantic Parsing: A Unified Framework and A Text-to-SQL Case Study

Ziyu Yao[1], Yu Su[1], Huan Sun[1], Wen-tau Yih[2]*
{yao.470, su.809, sun.397}@osu.edu
scottyih@fb.com
[1]The Ohio State University
[2]Facebook AI Research, Seattle

Abstract

As a promising paradigm, *interactive semantic parsing* has shown to improve both semantic parsing accuracy and user confidence in the results. In this paper, we propose a new, unified formulation of the interactive semantic parsing problem, where the goal is to design a *model-based* intelligent agent. The agent maintains its own state as the current predicted semantic parse, decides whether and where human intervention is needed, and generates a clarification question in natural language. A key part of the agent is a world model: it takes a percept (either an initial question or subsequent feedback from the user) and transitions to a new state. We then propose a simple yet remarkably effective instantiation of our framework, demonstrated on two text-to-SQL datasets (WikiSQL and Spider) with different state-of-the-art base semantic parsers. Compared to an existing interactive semantic parsing approach that treats the base parser as a black box, our approach solicits less user feedback but yields higher run-time accuracy.[1]

1 Introduction

Natural language interfaces that allow users to query data and invoke services without programming have been identified as a key application of semantic parsing (Berant et al., 2013; Thomason et al., 2015; Dong and Lapata, 2016; Zhong et al., 2017; Campagna et al., 2017; Su et al., 2017). However, existing semantic parsing technologies often fall short when deployed in practice, facing several challenges: (1) user utterances can be inherently ambiguous or vague, making it difficult to get the correct result in one shot, (2) the accuracy of state-of-the-art semantic parsers are still not high enough for real use, and (3) it is hard for users to validate the semantic parsing results, especially with mainstream neural network models that are known for the lack of interpretability.

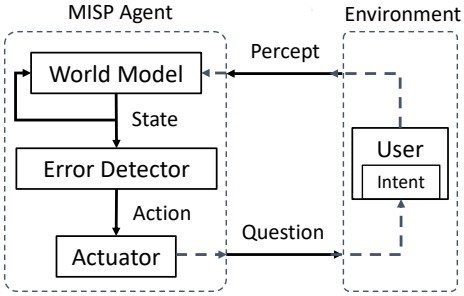


Figure 1: Model-based Interactive Semantic Parsing (MISP) framework.

In response to these challenges, *interactive semantic parsing* has been proposed recently as a practical solution, which includes human users in the loop to resolve utterance ambiguity, boost system accuracy, and improve user confidence via human-machine collaboration (Li and Jagadish, 2014; He et al., 2016; Chaurasia and Mooney, 2017; Su et al., 2018; Gur et al., 2018; Yao et al., 2019). For example, Gur et al. (2018) built the DialSQL system to detect errors in a generated SQL query and request user selection on alternative options via dialogues. Similarly, Chaurasia and Mooney (2017) and Yao et al. (2019) enabled semantic parsers to ask users clarification questions while generating an If-Then program. Su et al. (2018) showed that users overwhelmingly preferred an interactive system over the non-interactive counterpart for natural language interfaces to web APIs. While these recent studies successfully demonstrated the value of interactive semantic parsing in practice, they are often bound to a certain type of formal language or dataset, and the designs are thus ad-hoc and not easily generalizable. For example, DialSQL only applies

*Work started while at AI2
[1]Code available at https://github.com/sunlab-osu/MISP.

to SQL queries on the WikiSQL dataset (Zhong et al., 2017), and it is non-trivial to extend it to other formal languages (e.g., λ-calculus) or even just to more complex SQL queries beyond the templates used to construct the dataset.

Aiming to develop a general principle for building interactive semantic parsing systems, in this work we propose model-based interactive semantic parsing (MISP), where the goal is to design a *model-based intelligent agent* (Russell and Norvig, 2009) that can interact with users to complete a semantic parsing task. Taking an utterance (e.g., a natural language question) as input, the agent forms the semantic parse (e.g., a SQL query) in steps, potentially soliciting user feedback in some steps to correct parsing errors. As illustrated in Figure 1, a MISP agent maintains its *state* as the current semantic parse and, via an *error detector*, decides whether and where human intervention is needed (the *action*). This action is performed by a *question generator* (the *actuator*), which generates and presents to the user a human-understandable question. A core component of the agent is a *world model* (Ha and Schmidhuber, 2018) (hence *model-based*), which incorporates user feedback from the environment and transitions to a new agent state (e.g., an updated semantic parse). This process repeats until a terminal state is reached. Such a design endows a MISP agent with three crucial properties of interactive semantic parsing: (1) being *introspective* of the reasoning process and knowing when it may need human supervision, (2) being able to *solicit user feedback* in a human-friendly way, and (3) being able to *incorporate user feedback* (through state transitions controlled by the world model).

The MISP framework provides several advantages for designing an interactive semantic parser compared to the existing ad-hoc studies. For instance, the whole problem is conceptually reduced to building three key components (i.e., the world model, the error detector, and the actuator), and can be handled and improved separately. While each component may need to be tailored to the specific task, the general framework remains unchanged. In addition, the formulation of a model-based intelligent agent can facilitate the application of other machine learning techniques like reinforcement learning.

To better demonstrate the advantages of the MISP framework, we propose a simple yet remarkably effective instantiation for the text-to-SQL task. We show the effectiveness of the framework based on three base semantic parsers (SQLNet, SQLova and SyntaxSQLNet) and two datasets (WikiSQL and Spider). We empirically verified that with a small amount of targeted, test-time user feedback, interactive semantic parsers improve the accuracy by 10% to 15% absolute. Compared to an existing interactive semantic parsing system, DialSQL (Gur et al., 2018), our approach, despite its much simpler yet more general system design, achieves better parsing accuracy by asking only half as many questions.

2 Background & Related Work

Semantic Parsing. Mapping natural language utterances to their formal semantic representations, semantic parsing has a wide range of applications, including question answering (Berant et al., 2013; Dong and Lapata, 2016; Finegan-Dollak et al., 2018), robot navigation (Artzi and Zettlemoyer, 2013; Thomason et al., 2015) and Web API calling (Quirk et al., 2015; Su et al., 2018). The target application in this work is text-to-SQL, which has been popularized by the WikiSQL dataset (Zhong et al., 2017). One of the top-performing models on WikiSQL is SQLNet (Xu et al., 2017), which leverages the pre-defined SQL grammar sketches on WikiSQL and solves the SQL generation problem via "slot filling." By augmenting SQLNet with a table-aware BERT encoder (Devlin et al., 2019) and by revising the value prediction in `WHERE` clauses, SQLova (Hwang et al., 2019) advances further the state of the art. Contrast to WikiSQL, the recently released Spider dataset (Yu et al., 2018c) focuses on complex SQL queries containing multiple keywords (e.g., `GROUP BY`) and may join multiple tables. To handle such complexity, Yu et al. (2018b) proposed SyntaxSQLNet, a syntax tree network with modular decoders, which generates a SQL query by recursively calling a module following the SQL syntax. However, because of the more realistic and challenging setting in Spider, it only achieves 20% in accuracy.

We experiment our MISP framework with the aforementioned three semantic parsers on both WikiSQL and Spider. The design of MISP allows naturally integrating them as the base parser. For example, when SQLNet fills a sequence of slots to produce a SQL query, a "state" in MISP corresponds to a partially generated SQL query and it

transitions as SQLNet fills the next slot.

Interactive Semantic Parsing. To enhance parsing accuracy and user confidence in practical applications, interactive semantic parsing has emerged as a promising solution (Li and Jagadish, 2014; He et al., 2016; Chaurasia and Mooney, 2017; Su et al., 2018; Gur et al., 2018; Yao et al., 2019). Despite their effectiveness, existing solutions are somewhat ad-hoc and bound to a specific formal language and dataset. For example, DialSQL (Gur et al., 2018) is curated for WikiSQL, where SQL queries all follow the same and given grammar sketch. Similarly, (Yao et al., 2019) relies on a pre-defined two-level hierarchy among components in an If-Then program and cannot generalize to formal languages with a deeper structure. In contrast, MISP aims for a general design principle by explicitly identifying and decoupling important components, such as error detector, question generator and world model. It also attempts to integrate and leverage a strong base semantic parser, and transforms it to a natural interactive semantic parsing system, which substantially reduces the engineering cost.

3 Model-based Interactive Semantic Parsing

We now discuss the MISP framework (Figure 1) in more detail. Specifically, we highlight the function of each major building block and the relationships among them, and leave the description of a concrete embodiment to Section 4.

Environment. The *environment* consists of a user with a certain intent, which corresponds to a semantic parse that the user expects the agent to produce. Based on this intent, the user gives an initial natural language utterance u_0 to start a semantic parsing *session* and responds to any clarification question from the agent with feedback u_t at interaction turn t.

Agent State. The *agent state* s is an agent's internal interpretation of the environment based on all the available information. A straightforward design of the agent state is as the currently predicted semantic parse. It can also be endowed with meta information of the parsing process such as prediction probability or uncertainty to facilitate error detection.

World Model. A key component of a MISP agent is its *world model* (Ha and Schmidhuber, 2018), which compresses the historical percepts throughout the interaction and predicts the future based on the agent's knowledge of the world. More specifically, it models the transition of the agent state, $p(s_{t+1}|s_t, u_t)$, where u_t is the user feedback at step t and s_{t+1} is the new state. The transition can be deterministic or stochastic.

Error Detector. A MISP agent introspects its state and decides whether and where human intervention is needed. The *error detector* serves this role. Given the current state s_t (optionally the entire interaction history) and a set of *terminal states*, it decides on an *action* a_t: If the agent is at a terminal state, it terminates the session, executes the semantic parse, and returns the execution results to the user; otherwise, it determines a span in the current semantic parse that is likely erroneous and passes it, along with necessary context information needed to make sense of the error span, to the actuator.

Actuator. An *actuator* has a user-facing interface and realizes an agent's actions in a user-friendly way. In practice, it can be a natural language generator (NLG) (He et al., 2016; Gur et al., 2018; Yao et al., 2019) or an intuitive graphical user interface (Su et al., 2018; Berant et al., 2019), or the two combined.

4 MISP-SQL: An Instantiation of MISP for Text-to-SQL

Under the MISP framework, we design an interactive semantic parsing system (Figure 2), named MISP-SQL, for the task of text-to-SQL translation. MISP-SQL assumes a base text-to-SQL parser and leverages it to design the world model and the error detector. The world model is essentially a wrapper that takes the user input and changes the behavior of the base semantic parser (e.g., by changing the probability distribution or removing certain prediction paths). The error detector makes decisions based on the *uncertainty* of the predictions: if the parser is uncertain about a prediction, it is more likely to be an error. The actuator is a template-based natural language question generator developed for the general SQL language. Figure 2 shows an example of the MISP-SQL agent.

4.1 Agent State

For ease of discussion, we assume the base parser generates the SQL query by predicting a sequence

of SQL components,[2] as in many state-of-the-art systems (Xu et al., 2017; Wang et al., 2018; Yu et al., 2018a; Hwang et al., 2019). Agent state s_t is thus defined as a *partial* SQL query, i.e., s_t=$\{o_1, o_2, ..., o_t\}$, where o_t is the predicted SQL component at time step t, such as `SELECT place` in Figure 2. What constitutes a SQL component is often defined differently in different semantic parsers, but typically dictated by the SQL syntax. To support introspection and error detection, each prediction is associated with its uncertainty, which is discussed next.

4.2 Error Detector

The error detector in MISP-SQL is introspective and greedy. It is introspective because it examines the uncertainty of the predictions as opposed to the predictions themselves. On the other hand, it is greedy because its decisions are solely based on the last prediction o_t instead of the entire state s_t.

We experiment with two uncertainty measures, based on the probability of o_t estimated by the base semantic parser, as well as its standard deviation under Bayesian dropout (Gal and Ghahramani, 2016), respectively.

Probability-based Uncertainty. Intuitively if the base semantic parser gives a low probability to the top prediction at a step, it is likely uncertain about the prediction. Specifically, we say a prediction o_t needs user clarification if its probability is lower than a threshold p^*, i.e.,

$$p(o_t) < p^*.$$

This strategy is shown to be strong in detecting misclassified and out-of-distribution examples (Hendrycks and Gimpel, 2017).

Dropout-based Uncertainty. Dropout (Srivastava et al., 2014) has been used as a Bayesian approximation for estimating model uncertainty (Gal and Ghahramani, 2016) in several tasks (Dong et al., 2018; Siddhant and Lipton, 2018; Xiao and Wang, 2019). Different from its standard application to prevent models from overfitting in training time, we use it at test time to measure model uncertainty, similar to (Dong et al., 2018). The intuition is that if the probability on a prediction varies dramatically (as measured by the standard deviation) across different perturbations under dropout, the model is likely uncertain about it. Specifically, the uncertainty on prediction o_t is calculated as:

$$\text{STDDEV}\{p(o_t|W_i)\}_{i=1}^{N},$$

where W_i is the parameters of the base semantic parser under the i-th dropout perturbation, and the uncertainty score is the standard deviation of the prediction probabilities over N random passes. We say o_t needs user clarification if its uncertainty score is greater than a threshold s^*.

Terminal State. The only terminal state is when the base semantic parser indicates end of parsing.

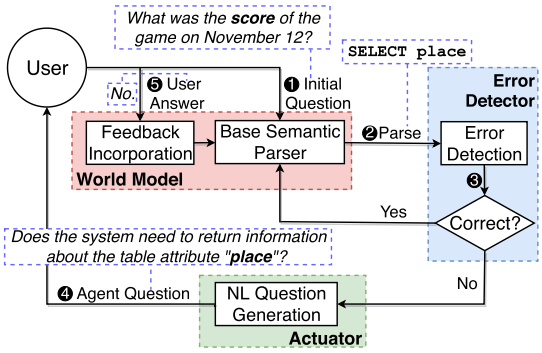


Figure 2: MISP-SQL Agent. The base semantic parser incrementally parses the user question (Step 1) into a SQL query by first selecting a column from the table (Step 2). This partial parse is examined by the error detector (Step 3), who determines that the prediction is incorrect (because the uncertainty is high) and triggers the actuator to ask the user a clarification question (Step 4). The user feedback is then incorporated into the world model (Step 5) to update the agent state. If the prediction was correct, Step 2 would be repeated to continue the parsing.

4.3 Actuator: An NL Generator

The MISP-SQL agent performs its action (e.g., validating the column "*place*") via asking users *binary* questions, hence the actuator is a natural language generator (NLG). Although there has been work on describing a SQL query with an NL statement (Koutrika et al., 2010; Ngonga Ngomo et al., 2013; Iyer et al., 2016; Xu et al., 2018), few work studies generating *questions* about a certain SQL component in a systematic way.

Inspired by (Koutrika et al., 2010; Wang et al., 2015), we define a rule-based NLG, which consists of a seed lexicon and a grammar for deriving questions. Table 1 shows rules covering

[2] In practice this assumption may not be necessary as long as there is a reasonable way to chunk the semantic parse to calculate uncertainty and formulate clarification questions.

[Lexicon]
is greater than|equals to|is less than → OP[>|=|<]
sum of values in|average value in|number of|minimum value in|maximum value in → AGG[`sum`|`avg`|`count`|`min`|`max`]
[Grammar]
"*col*" → COL[*col*]
Does the system need to return information about COL[*col*] ? → Q[*col*‖`SELECT` *agg*? *col*]
Does the system need to return AGG[*agg*] COL[*col*] ? → Q[*agg*‖`SELECT` *agg col*]
Does the system need to return a value after any mathematical calculations on COL[*col*] ? → Q[*agg*=None‖`SELECT` *col*]
Does the system need to consider any conditions about COL[*col*] ? → Q[*col*‖`WHERE` *col op val*]
The system considers the following condition: COL[*col*] OP[*op*] a value. Is this condition correct? → Q[*op*‖`WHERE` *col op val*]
The system considers the following condition: COL[*col*] OP[*op*] *val*. Is this condition correct? → Q[*val*‖`WHERE` *col op val*]

Table 1: Domain-general lexicon and grammar for NL generation in MISP-SQL (illustrated for WikiSQL; a more comprehensive grammar for Spider can be found in Appendix A).

SQL queries on WikiSQL (Zhong et al., 2017). The seed lexicon defines NL descriptions for basic SQL elements in the form of "$n \rightarrow t[p]$", where n is an NL phrase, t is a pre-defined syntactic category and p is either an aggregator (e.g., `avg`) or an operator (e.g., $>$). For example, "is greater than $\rightarrow$ OP[$>$]" specifies a phrase "is greater than" to describe the operator "$>$". In MISP-SQL, we consider four syntactic categories: AGG for aggregators, OP for operators, COL for columns and Q for generated questions. However, it can be extended with more lexicon entries and grammar rules to accommodate more complex SQL in Spider (Yu et al., 2018c), which we show in Appendix A.

The grammar defines rules to derive questions. Each column is described by itself (i.e., the column name). Rules associated with each Q-typed item "Q[v‖`Clause`]" constructs an NL question asking about v in `Clause`. The `Clause` is the necessary context to formulate meaningful questions. Figure 3 shows a derivation example. Note that, both the lexicon and the grammar in our system are domain-agnostic in the sense that it is not specific to any database. Therefore, it can be reused for new domains in the future. Database-specific rules, such as naming each column with a more canonical phrase (rather than the column name), are also possible.

4.4 World Model

The agent incorporates user feedback and updates its state with a world model. Different from DialSQL which trains an additional neural network, the MISP-SQL agent directly employs the base semantic parser to transition states, which saves additional training efforts.

As introduced in Section 4.3, the agent raises a binary question to the user about a predicted SQL component o_t. Therefore, the received user feedback either confirms the prediction or negates it. In the former case, the state is updated by proceeding to the next decoding step, i.e., s_{t+1}={o_1, ..., o_t, o_{t+1}}, where o_{t+1} is the predicted next component and s_{t+1} shows the updated partial parse. In the latter case, the user feedback is incorporated to constrain the search space of the base parser (i.e., forbidding the parser from making the same wrong prediction), based on which the parser refreshes its prediction and forms a new state s_{t+1}={o_1, ..., o_{t-1}, o_{t+1}}, where o_{t+1} is a predicted alternative to replace o_t. To avoid being trapped in a large search space, for each SQL component, we consider a maximum number of K alternatives (in addition to the original prediction) to solicit user feedback on.

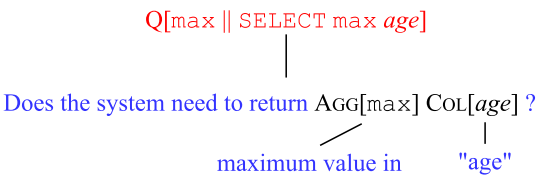


Figure 3: Deriving an NL question about the aggregator `max` in the clause "`SELECT max`(age)" from the rooted Q-typed item .

5 Experiments

We apply our approach to the task of mapping natural language questions to SQL queries. In this section, we first describe the basic setup, including the datasets and the base semantic parsers, followed by the system results on both simulated and real users.

5.1 Experimental Setup

We evaluate our proposed MISP-SQL agent on WikiSQL (Zhong et al., 2017), which contains 80,654 hand-annotated pairs of ⟨NL question,

System	SQLNet			SQLova		
	Acc_{qm}	**Acc_{ex}**	**Avg. #q**	**Acc_{qm}**	**Acc_{ex}**	**Avg. #q**
no interaction	0.615	0.681	N/A	0.797	0.853	N/A
DialSQL	0.690	N/A	2.4	N/A	N/A	N/A
MISP-SQL$^{\text{Unlimit10}}$	0.932	0.948	7.445	0.985	0.991	6.591
MISP-SQL$^{\text{Unlimit3}}$	0.870	0.900	7.052	0.955	0.974	6.515
MISP-SQL$^{p^*=0.95}$	0.782	0.824	1.713	0.912	0.939	0.773
MISP-SQL$^{p^*=0.8}$	0.729	0.779	1.104	0.880	0.914	0.488

Table 2: Simulation evaluation of MISP-SQL (based on SQLNet or SQLova) on WikiSQL Test set. "MISP-SQL$^{p^*=\text{X}}$" denotes our agent with probability-based error detection (threshold at X). "MISP-SQL$^{\text{UnlimitK}}$" denotes a variant that asks questions for every component, with up to $K+1$ questions per component.

SQL query$\rangle$, distributed across 24,241 tables from Wikipedia. Our experiments follow the same data split as in (Zhong et al., 2017).

We experiment MISP-SQL with two base semantic parsers: SQLNet (Xu et al., 2017) and SQLova (Hwang et al., 2019). Unlike in DialSQL's evaluation (Gur et al., 2018), we do not choose Seq2SQL (Zhong et al., 2017) as a base parser but SQLova instead, because it achieves similar performance as SQLNet while SQLova is currently the best open-sourced model on WikiSQL, which can give us a more comprehensive evaluation. For each of the two base semantic parsers, we test our agent with two kinds of error detectors, based on prediction probability and Bayesian dropout, respectively (Section 4.2). We tune the threshold p^* within $0.5 \sim 0.95$ and s^* within $0.01 \sim 0.2$. Particularly for uncertainty-based detection measured by Bayesian dropout, the number of passes N is set to 10, with a dropout rate 0.1. The dropout layers are applied at the same positions as when each semantic parser is trained. When the agent interacts with users, the maximum number of alternative options (in addition to the original prediction) per component, K, is set to 3. If the user negates all the $K+1$ predicted candidates, the agent will keep the original prediction, as in (Gur et al., 2018).

5.2 Simulation Evaluation

In simulation evaluation, each agent interacts with a *simulated* user, who gives a yes/no answer based on the ground-truth SQL query. If the agent fails to correct its predictions in three consecutive interaction turns, the user will leave the interaction early and the agent has to finish the remaining generation without further help from the user.

Overall Comparison. We first compare MISP-SQL with the two base semantic parsers without interactions in Table 2. For SQLNet, we also compare our system with the reported performance of DialSQL (Gur et al., 2018, Table 4). However, since DialSQL is not open-sourced and it is not easy to reproduce it, we are unable to adapt it to SQLova for more comparisons. Following (Xu et al., 2017; Hwang et al., 2019), we evaluate the SQL query match accuracy ("**Acc_{qm}**", after converting the query into its canonical form) and the execution accuracy ("**Acc_{ex}**") of each agent. "**Avg. #q**" denotes the average number of questions per query. For any base parser, MISP-SQL improves their performance by interacting with users. Particularly for SQLNet, MISP-SQL outperforms the DialSQL system with only *half* the number of questions (1.104 vs. 2.4), and has a much simpler design without the need of training an extra model (besides training the base parser, which DialSQL needs to do as well). Our agent can even boost the strong performance of SQLova from 85% to 94% in execution accuracy, with merely 0.773 questions per query.

We also present an "upper-bounded" accuracy of our agent, when it does not adopt any error detector and asks questions about *every* component with at most 10 ("MISP-SQL$^{\text{Unlimit10}}$") or 3 ("MISP-SQL$^{\text{Unlimit3}}$") alternatives. Interestingly, even for the weaker SQLNet parser, most true predictions have already been contained within the top 10 options (giving 0.932 query match accuracy). When equipped with the stronger SQLova parser, the agent has a potential to boost the execution accuracy to around 100% by considering only the top 3 options of every prediction. The complete results can be found in Appendix B.

Error Detector Comparison. We then compare the probability-based and dropout-based error detectors in Figure 4, where each marker indicates

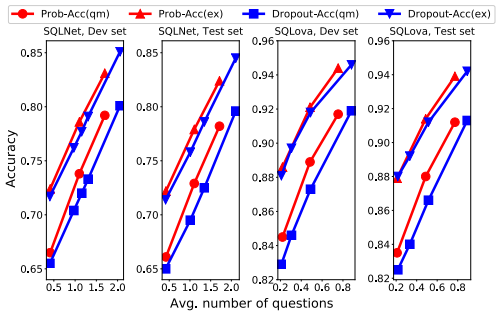


Figure 4: Comparison of probability- and dropout-based error detection.

the agent's accuracy and the average number of questions it needs under a certain error detection threshold. Consistently for both SQLNet and SQLova, the probability-based error detector can achieve the same accuracy with a lower number of questions than the dropout-based detector. It is also observed that this difference is greater in terms of query match accuracy, around 0.15 $\sim$ 0.25 for SQLNet and 0.1 $\sim$ 0.15 for SQLova. A more direct comparison of various settings under the same average number of questions can be found in Appendix C.

To better understand how each kind of error detectors works, we investigate the portion of questions that each detector spends on *right* predictions (denoted as "$\mathbf{Q_r}$"). An ideal system should ask fewer questions on right predictions while identify more truly incorrect predictions to fix the mistakes. We present the question distributions of the various systems in Table 3. One important conclusion drawn from this table is that probability-based error detection is much more effective on identifying incorrect predictions. Consider the system using probability threshold 0.5 for error detection (i.e., "p^*=0.5") and the one using dropout-based error detector with a threshold 0.2 (i.e., "s^*=0.2") on SQLNet. When both systems ask around the same number of questions during the interaction, the former spends only 16.9% of unnecessary questions on correct predictions (Q_r), while the latter asks twice amount of them (32.1%). Similar situations are also observed for SQLova. It is also notable that, when the probability threshold is lower (which results in a fewer total number of questions), the portion of questions on right actions drops significantly (e.g., from 23.0% to 16.9% when the threshold changes from 0.8 to 0.5 on SQLNet). However, this portion remains almost unchanged for dropout-based error detection.

SQLNet			SQLova		
System	**Avg. #q**	**Q_r%**	**System**	**Avg. #q**	**Q_r%**
p^*=0.8	1.099	23.0%	p^*=0.8	0.484	28.9%
p^*=0.5	0.412	16.9%	p^*=0.5	0.220	18.4%
s^*=0.07	1.156	34.5%	s^*=0.03	0.489	50.4%
s^*=0.2	0.406	32.1%	s^*=0.05	0.306	52.5%

Table 3: Portion of interaction questions on right predictions (Q_r%) for each agent setting on WikiSQL Dev set (smaller is better). "p^*/s^*=X" denotes our agent with probability/dropout-based error detection (threshold at X).

5.3 Extend to Complex SQL Generation

A remarkable characteristic of MISP-SQL is its generalizability, as it makes the best use of the base semantic parser and requires no extra model training. To verify it, we further experiment MISP-SQL on the more complex text-to-SQL dataset "Spider" (Yu et al., 2018c). The dataset consists of 10,181 questions on multi-domain databases, where SQL queries can contain complex keywords such as `GROUP BY` or join several tables. We extend the NLG lexicon and grammar (Section 4.3) to accommodate this complexity, with details shown in Appendix A.

We adopt SyntaxSQLNet (Yu et al., 2018b) as the base parser.[3] In our experiments, we follow the same database split as in (Yu et al., 2018c) and report the Exact Matching accuracy ("$\mathbf{Acc_{em}}$") on Dev set.[4] Other experimental setups remain the same as when evaluating MISP-SQL on WikiSQL. Table 4 shows the results.

We first observe that, via interactions with simulated users, MISP-SQL improves SyntaxSQLNet by 10% accuracy with reasonably 3 questions per query. However, we also realize that, unlike on WikiSQL, in this setting, the probability-based error detector requires more questions than the Bayesian uncertainty-based detector. This can be explained by the inferior performance of the base SyntaxSQLNet parser (merely 20% accuracy without interaction). In fact, the portion of questions that the probability-based detector spends on right predictions (Q_r) is still half of that the dropout-based detector asks (12.8% vs. 24.8%). However, it wastes around 60% of questions on unsolvable wrong predictions. This typically hap-

[3] We chose SyntaxSQLNet because it was the best model by the paper submission time. In principle, our framework can also be applied to more sophisticated parsers such as (Bogin et al., 2019; Guo et al., 2019).

[4] We do not report results on Spider test set since it is not publicly available.

System	Acc_{em}	Avg. #q
no interaction	0.190	N/A
MISP-SQL$^{\text{Unlimit10}}$	0.522	14.878
MISP-SQL$^{\text{Unlimit3}}$	0.382	11.055
MISP-SQL$^{p^*=0.95}$	0.300	3.908
MISP-SQL$^{p^*=0.8}$	0.268	3.056
MISP-SQL$^{s^*=0.01}$	0.315	3.815
MISP-SQL$^{s^*=0.03}$	0.290	2.905

Table 4: Simulation evaluation of MISP-SQL (built on SyntaxSQLNet) on Spider Dev set.

pens when the base parser is not strong enough, i.e., cannot rank the true option close to the top, or when there are unsolved wrong precedent predictions (e.g., in "`WHERE` *col op val*", when *col* is wrong, whatever *op*/*val* following it is wrong). This issue can be alleviated when more advanced base parsers are adopted in the future.

5.4 Human Evaluation

We further conduct human user study to evaluate the MISP-SQL agent. Our evaluation setting largely follows Gur et al. (2018). For each base semantic parser, we randomly sample 100 examples from the corresponding dataset (either WikiSQL Test set or Spider Dev set) and ask three human evaluators, who are graduate students with only rudimentary knowledge of SQL based on our survey, to work on each example and then report the averaged results. We present to the evaluators the initial natural language question and allow them to view the table headers to better understand the question intent. On Spider, we also show the name of the database tables. We select error detectors based on the simulation results: For SQLNet and SQLova, we equip the agent with a probability-based error detector (threshold at 0.95); for SyntaxSQLNet, we choose a Bayesian uncertainty-based error detector (threshold at 0.03). As in the simulation evaluation, we cannot directly compare with DialSQL in human evaluation because the code is not yet publicly available.

Table 5 shows the results. In all settings, MISP-SQL improves the base parser's performance, demonstrating the benefit of involving human interaction. However, we also notice that the gain is not as large as in simulation, especially on SQLova. Through interviews with the human evaluators, we found that the major reason is that they sometimes had difficulties understanding the true intent of some test questions that are ambiguous, vague, or contain entities they are not familiar with. We believe this reflects a general challenge of setting up human evaluation for semantic parsing that is close to the real application setting, and thus set forth the following discussion.

System	$\text{Acc}_{\text{qm/em}}$	Acc_{ex}	Avg. #q
SQLNet			
no interaction	0.580	0.660	N/A
MISP-SQL (simulation)	0.770	0.810	1.800
MISP-SQL (real user)	0.633	0.717	1.510
SQLova			
no interaction	0.830	0.890	N/A
MISP-SQL (simulation)	0.920	0.950	0.550
MISP-SQL (real user)	0.837	0.880	0.533
+ w/ full info.	0.907	0.937	0.547
SyntaxSQLNet			
no interaction	0.180	N/A	N/A
MISP-SQL (simulation)	0.290	N/A	2.730
MISP-SQL (real user)	0.230	N/A	2.647

Table 5: Human evaluation on 100 random examples for MISP-SQL agents based on SQLNet, SQLova and SyntaxSQLNet, respectively.

5.5 Discussion on Future Human Evaluation

Most human evaluation studies for (interactive) semantic parsers so far (Chaurasia and Mooney, 2017; Gur et al., 2018; Su et al., 2018; Yao et al., 2019) use pre-existing test questions (e.g., from datasets like WikiSQL). However, this introduces an undesired discrepancy, that is, human evaluators may not necessarily be able to understand the true intent of the given questions in an faithful way, especially when the question is ambiguous, vague, or containing unfamiliar entities.

This discrepancy is clearly manifested in our human evaluation with SQLova (Table 5). When the base parser is strong, many of the remaining incorrectly parsed questions are challenging not only for the base parser but also for human evaluators. We manually examined the situations where evaluators made a different choice than the simulator and found that 80% of such choices happened when the initial question is ambiguous or the gold SQL annotation is wrong. For example, for the question "*name the city for kanjiža*" it is unlikely for human evaluators to know that "kanjiža" is an "Urban Settlement" without looking at the table content or knowing the specific background knowledge beforehand. This issue has also been reported as the main limitation to further improve SQLova (Hwang et al., 2019), which

could in principle be resolved by human interactions if the users have a clear and consistent intent in mind. To verify this, we conduct an additional experiment with SQLova where human evaluators can view the table content as well as the gold SQL query before starting the interaction to better understand the true intent (denoted as "w/ full info" in Table 5). As expected, the MISP-SQL agent performs much better (close to simulation) when users know what they are asking. It further confirms that a non-negligible part of the accuracy gap between simulation and human evaluation is due to human evaluators not fully understanding the question intent and giving false feedback.

To alleviate this discrepancy, a common practice is to show human evaluators the schema of the underlying database, as Gur et al. (2018) and we did (Section 5.4), but it is still insufficient, especially for entity-related issues (e.g., "kanjiža"). On the other hand, while exposing human evaluators to table content helps resolve the entity-related issues, it is likely to introduce undesired biases in favor of the system under test (i.e., "overexposure"), since human evaluators may then be able to give more informative feedback than real users.

To further reduce the discrepancy between human evaluation and real use cases, one possible solution is to ask human evaluators to come up with questions from scratch (instead of using pre-existing test questions), which guarantees intent understanding. While this solution may still require exposure of table content to evaluators (such that they can have some sense of each table attribute), overexposure can be mitigated by showing them only part (e.g., just a few rows) of the table content, similar to the annotation strategy by Zhong et al. (2017). Furthermore, the reduced controllability on the complexity of the evaluator-composed questions can be compensated by conducting human evaluation in a larger scale. We plan to explore this setting in future work.

6 Conclusion and Future Work

This work proposes a new and unified framework for the interactive semantic parsing task, named MISP, and instantiates it successfully on the text-to-SQL task. We outline several future directions to further improve MISP-SQL and develop MISP systems for other semantic parsing tasks:

Improving Agent Components. The flexibility of MISP allows improving on each agent component separately. Take the error detector for example. One can augment the probability-based error detector in MISP-SQL with probability calibration, which has been shown useful in aligning model confidence with its reliability (Guo et al., 2017). One can also use learning-based approaches, such as a reinforced decision policy (Yao et al., 2019), to increase the rate of identifying wrong and solvable predictions.

Lifelong Learning for Semantic Parsing. Learning from user feedback is a promising solution for lifelong semantic parser improvement (Iyer et al., 2017; Padmakumar et al., 2017; Labutov et al., 2018). However, this may lead to a non-stationary environment (e.g., changing state transition) from the perspective of the agent, making its training (e.g., error detector learning) unstable. In the context of dialog systems, Padmakumar et al. (2017) suggests that this effect can be mitigated by jointly updating the dialog policy and the semantic parser batchwisely. We leave exploring this aspect in our task to future work.

Scaling Up. It is important for MISP agents to scale to larger backend data sources (e.g., knowledge bases like Freebase or Wikidata). To this end, one can improve MISP from at least three aspects: (1) using more intelligent interaction designs (e.g., free-form text as user feedback) to speed up the hypothesis space searching globally, (2) strengthening the world model to nail down a smaller set of plausible hypotheses based on both the initial question and user feedback, and (3) training the agent to learn to improve the parsing accuracy while minimizing the number of required human interventions over time.

Acknowledgments

This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, Fujitsu gift grant, and Ohio Supercomputer Center (Center, 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Jonathan Berant, Daniel Deutch, Amir Globerson, Tova Milo, and Tomer Wolfson. 2019. Explaining queries over web tables to non-experts. In *Proceedings of the 35th IEEE International Conference on Data Engineering (ICDE)*.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-SQL parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565.

Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S Lam. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *Proceedings of the 26th International Conference on World Wide Web*, pages 341–350. International World Wide Web Conferences Steering Committee.

Ohio Supercomputer Center. 1987. Ohio supercomputer center. http://osc.edu/ark:/19495/f5s1ph73.

Shobhit Chaurasia and Raymond J Mooney. 2017. Dialog for language to code. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 175–180.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 33–43.

Li Dong, Chris Quirk, and Mirella Lapata. 2018. Confidence modeling for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 743–753.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1339–1349.

David Ha and Jürgen Schmidhuber. 2018. World models. *ArXiv preprint arXiv:1803.10122*.

Luheng He, Julian Michael, Mike Lewis, and Luke Zettlemoyer. 2016. Human-in-the-loop parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2337–2342.

Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proceedings of International Conference on Learning Representations*.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on WikiSQL with table-aware word contextualization. *ArXiv preprint arXiv:1902.01069*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2073–2083.

Georgia Koutrika, Alkis Simitsis, and Yannis E Ioannidis. 2010. Explaining structured queries in natural language. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 333–344. IEEE.

Igor Labutov, Bishan Yang, and Tom Mitchell. 2018. Learning to learn semantic parsers from natural language supervision. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1676–1690.

Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. 2013. Sorry, I don't speak SPARQL: translating SPARQL queries into natural language. In *Proceedings of the 22nd international conference on World Wide Web*, pages 977–988. ACM.

Aishwarya Padmakumar, Jesse Thomason, and Raymond J Mooney. 2017. Integrated learning of dialog strategies and semantic parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 547–557.

Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to code: Learning semantic parsers for If-This-Then-That recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 878–888.

Stuart Russell and Peter Norvig. 2009. Artificial intelligence: A modern approach.

Aditya Siddhant and Zachary C Lipton. 2018. Deep Bayesian active learning for natural language processing: Results of a large-scale empirical study. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2904–2909.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. 2017. Building natural language interfaces to Web APIs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 177–186. ACM.

Yu Su, Ahmed Hassan Awadallah, Miaosen Wang, and Ryen W White. 2018. Natural language interfaces with fine-grained user interaction: A case study on web APIs. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Jesse Thomason, Shiqi Zhang, Raymond J Mooney, and Peter Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. 2018. Pointing out SQL queries from text.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1332–1342.

Yijun Xiao and William Yang Wang. 2019. Quantifying uncertainties in natural language processing tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7322–7329.

Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. 2018. SQL-to-text generation with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 931–936.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: Generating structured queries from natural language without reinforcement learning. *ArXiv preprint arXiv:1711.04436*.

Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, and Huan Sun. 2019. Interactive semantic parsing for if-then recipes via hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2547–2554.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018b. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018c. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *ArXiv preprint arXiv:1709.00103*.

A Extension to Complex SQL

Table 8 shows the extended lexicon entries and grammar rules in NLG for applying our MISP-SQL agent to generate more complex SQL queries, such as those on Spider (Yu et al., 2018c). In this dataset, a SQL query can associate with multiple tables. Therefore, we name a column by combining the column name with its table name (i.e., "col" in table "tab" $\rightarrow$ COL[col (table tab)]). For simplicity, we omit "(table tab)" when referring to a column col in the grammar.

B Simulation Evaluation Results

The complete simulation experiment results of MISP-SQL agents (based on SQLNet and SQLova) are shown in Table 6 & 7.

C Error Detector Comparison

As a supplementary experiment to Figure 4, in this section, we show the performance of different error detectors under the same average number of questions ("*target budget*"). Specifically, for each base semantic parser and each kind of error detector, we tune its decision threshold (i.e., p^* and s^*) such that the resulting average number of questions ("*actual budget*") is as close to the target as possible. In practice, we relax the actual budget to be within ± 0.015 of the target budget, which empirically leads to merely negligible variance. The results are shown in Table 9-10 for SQLNet and Table 11-12 for SQLova.

	SQLNet		
System	**Acc_{qm}**	**Acc_{ex}**	**Avg. #q**
no interaction	0.615	0.681	N/A
MISP-SQL$^{\text{Unlimit10}}$	0.932	0.948	7.445
MISP-SQL$^{\text{Unlimit3}}$	0.870	0.900	7.052
MISP-SQL$^{p^*=0.95}$	0.782	0.824	1.713
MISP-SQL$^{p^*=0.8}$	0.729	0.779	1.104
MISP-SQL$^{p^*=0.5}$	0.661	0.722	0.421
MISP-SQL$^{s^*=0.01}$	0.796	0.845	2.106
MISP-SQL$^{s^*=0.05}$	0.725	0.786	1.348
MISP-SQL$^{s^*=0.1}$	0.695	0.758	1.009
MISP-SQL$^{s^*=0.2}$	0.650	0.714	0.413

Table 6: Simulation evaluation of MISP-SQL (based on SQLNet) on WikiSQL Test set.

	SQLova		
System	**Acc_{qm}**	**Acc_{ex}**	**Avg. #q**
no interaction	0.797	0.853	N/A
MISP-SQL$^{\text{Unlimit10}}$	0.985	0.991	6.591
MISP-SQL$^{\text{Unlimit3}}$	0.955	0.974	6.515
MISP-SQL$^{p^*=0.95}$	0.912	0.939	0.773
MISP-SQL$^{p^*=0.8}$	0.880	0.914	0.488
MISP-SQL$^{p^*=0.5}$	0.835	0.879	0.209
MISP-SQL$^{s^*=0.01}$	0.913	0.942	0.893
MISP-SQL$^{s^*=0.03}$	0.866	0.912	0.515
MISP-SQL$^{s^*=0.05}$	0.840	0.892	0.333
MISP-SQL$^{s^*=0.07}$	0.825	0.880	0.216

Table 7: Simulation evaluation of MISP-SQL (based on SQLova) on WikiSQL Test set.

	[Lexicon]
	is greater than (or equivalent to)|equals to|is less than (or equivalent to)|does not equal to → OP[> (=) |= |< (=) |! =]
	is IN|is NOT IN|follows a pattern like|is between → OP[in|not in|like|between]
	sum of values in|average value in|number of|minimum value in|maximum value in → AGG[sum|avg|count|min|max]
	in descending order (and limited to top N)|in ascending order (and limited to top N) → ORDER[desc (limit N)|asc (limit N)]
	[Grammar]
(R1)	"col" in table "tab" → COL[col (table tab)]
(R2)	Does the system need to return information about COL[col] ? → Q[col‖SELECT $agg?$ col]
(R3)	Does the system need to return AGG[agg] COL[col] ? → Q[agg‖SELECT agg col]
(R4)	Does the system need to return a value after any mathematical calculations on COL[col] ? → Q[agg=None‖SELECT agg col]
(R5)	Does the system need to consider any conditions about COL[col] ? → Q[col‖WHERE col op val]
(R6)	The system considers the following condition: COL[col] OP[op] a given literal value. Is this condition correct? → Q[$terminal$‖WHERE col op $terminal$]
(R7)	The system considers the following condition: COL[col] OP[op] a value to be calculated. Is this condition correct? → Q[$root$‖WHERE col op $root$]
(R8)	Do the conditions about COL[col_i] and COL[col_j] hold at the same time? → Q[AND‖WHERE col_i .. AND col_j ..]
(R9)	Do the conditions about COL[col_i] and COL[col_j] hold alternatively? → Q[OR‖WHERE col_i .. OR col_j ..]
(R10)	Does the system need to group items in table tab based on COL[col] before doing any mathematical calculations? → Q[col‖GROUP BY col]
(R11)	Given that the system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, does the system need to consider any conditions about COL[col] ? → Q[col‖GROUP BY col^g HAVING $agg?$ col]
(R12)	Given that the system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, does the system need to consider any conditions about AGG[agg] COL[col] ? → Q[agg‖GROUP BY col^g HAVING agg col]
(R13)	Given that the system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, does the system need to consider a value after any mathematical calculations on COL[col] ? → Q[agg=None‖GROUP BY col^g HAVING agg col]
(R14)	The system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, then considers the following condition: COL[col] OP[op] a value. Is this condition correct? → Q[op‖GROUP BY col^g HAVING $agg?$ col op val]
(R15)	Given that the system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, does it need to consider any conditions? → Q[NONE_HAVING ‖GROUP BY col^g NONE_HAVING]
(R16)	Does the system need to order results based on COL[col] ? → Q[col‖ORDER BY $agg?$ col]
(R17)	Does the system need to order results based on AGG[agg] COL[col] ? → Q[agg‖ORDER BY agg col]
(R18)	Does the system need to order results based on a value after any mathematical calculations on COL[col] ? → Q[agg=None‖ORDER BY agg col]
(R19)	Given that the system orders the results based on (AGG[agg]) COL[col], does it need to be ORDER[od] ? → Q[od‖ORDER BY $agg?$ col od]

Table 8: Extended lexicon and grammar for MISP-SQL NLG module to handle complex SQL on Spider.

Avg. #q	Probability-based		Dropout-based	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
0.5	0.672	0.732	0.663	0.726
1.0	0.725	0.775	0.706	0.765
1.5	0.778	0.820	0.749	0.809
2.0	0.812	0.848	0.796	0.845

Table 9: Comparison of error detectors for SQLNet with a target average number of questions on WikiSQL Dev set.

Avg. #q	Probability-based		Dropout-based	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
0.5	0.669	0.729	0.656	0.720
1.0	0.722	0.773	0.695	0.758
1.5	0.765	0.810	0.740	0.801
2.0	0.805	0.844	0.790	0.842

Table 10: Comparison of error detectors for SQLNet with a target average number of questions on WikiSQL Test set.

Avg. #q	Probability-based		Dropout-based	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
0.2	0.844	0.885	0.829	0.881
0.4	0.876	0.910	0.856	0.905
0.6	0.902	0.932	0.887	0.927
0.8	0.921	0.947	0.913	0.941

Table 11: Comparison of error detectors for SQLova with a target average number of questions on WikiSQL Dev set.

Avg. #q	Probability-based		Dropout-based	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
0.2	0.832	0.877	0.823	0.878
0.4	0.865	0.902	0.851	0.901
0.6	0.895	0.926	0.881	0.922
0.8	0.915	0.941	0.904	0.936

Table 12: Comparison of error detectors for SQLova with a target average number of questions on WikiSQL Test set.

An Imitation Game for Learning Semantic Parsers from User Interaction

Ziyu Yao[1], Yiqi Tang[1], Wen-tau Yih[2], Huan Sun[1], Yu Su[1]
{yao.470, tang.1466, sun.397, su.809}@osu.edu
scottyih@fb.com
[1]The Ohio State University
[2]Facebook AI Research, Seattle

Abstract

Despite the widely successful applications, building a semantic parser is still a tedious process in practice with challenges from costly data annotation and privacy risks. We suggest an alternative, human-in-the-loop methodology for learning semantic parsers directly from users. A semantic parser should be introspective of its uncertainties and prompt for user demonstrations when uncertain. In doing so it also gets to imitate the user behavior and continue improving itself autonomously with the hope that eventually it may become as good as the user in interpreting their questions. To combat the sparsity of demonstrations, we propose a novel *annotation-efficient imitation learning* algorithm, which iteratively collects new datasets by mixing demonstrated states and confident predictions and retrains the semantic parser in a Dataset Aggregation fashion (Ross et al., 2011). We provide a theoretical analysis of its cost bound and also empirically demonstrate its promising performance on the text-to-SQL problem.[1]

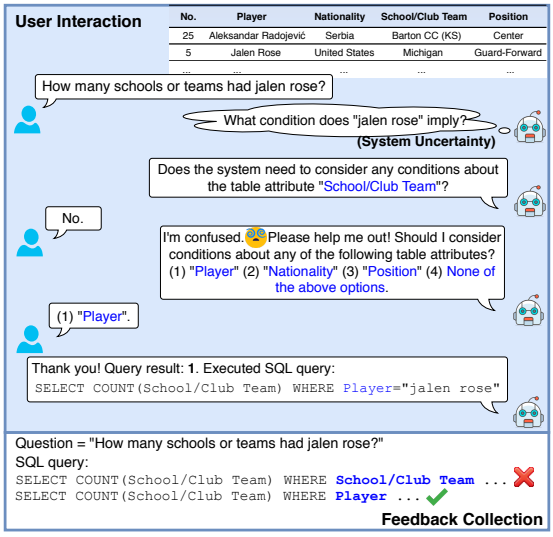


Figure 1: A semantic parser proactively interacts with the user in a friendly way to resolve its uncertainties. In doing so it also gets to imitate the user behavior and continue improving itself autonomously with the hope that eventually it may become as good as the user in interpreting their questions.

1 Introduction

Semantic parsing has found tremendous applications in building natural language interfaces that allow users to query data and invoke services without programming (Woods, 1973; Zettlemoyer and Collins, 2005; Berant et al., 2013; Yih et al., 2015; Su et al., 2017; Yu et al., 2018). The life cycle of a semantic parser typically consists of two stages: (1) *bootstrapping*, where we keep collecting labeled data via trained annotators and/or crowdsourcing for model training until it reaches commercial-grade performance (e.g., 95% accuracy on a surrogate test set), and (2) *fine-tuning*, where we deploy the system, analyze the usage, and collect and annotate new data to address the identified problems or emerging needs. However, it poses several challenges for scaling up or building semantic parsers for new domains: (1) *high bootstrapping cost* because mainstream neural parsing models are data-hungry and the annotation cost of semantic parsing data is relatively high, (2) *high fine-tuning cost* from continuously analyzing usage and annotating new data, and (3) *privacy risks* arising from exposing private user data to annotators and developers (Lomas, 2019).

In this paper, we suggest an alternative methodology for building semantic parsers that could potentially address all the aforementioned problems. The key is to involve human users in the learning loop. A semantic parser should be introspective of its uncertainties (Dong et al., 2018) and proac-

[1]Code will be available at https://github.com/sunlab-osu/MISP.

tively prompt for demonstrations from the user, who knows the question best, to resolve them. In doing so, the semantic parser can accumulate targeted training data and continue improving itself autonomously without involving any annotators or developers, hence also minimizing privacy risks. The bootstrapping cost could also be significantly reduced because an interactive system needs not to be almost perfectly accurate to be deployed. On the other hand, such interaction opens up the black box and allows users to know more about the reasoning underneath the system and better interpret the final results (Su et al., 2018). A human-in-the-loop methodology like this also opens the door for domain adaptation and personalization.

This work builds on the recent line of research on interactive semantic parsing (Li and Jagadish, 2014; Chaurasia and Mooney, 2017; Gur et al., 2018; Yao et al., 2019b). Specifically, Yao et al. (2019b) provide a general framework, MISP (Model-based Interactive Semantic Parsing), which handles uncertainty modeling and natural language generation. We will leverage MISP for user interaction to prove the feasibility of the envisioned methodology. However, existing studies only focus on interacting with users to resolve uncertainties. None of them has fully addressed the crucial problem of *how to continually learn from user interaction*, which is the technical focus of this study.

One form of user interaction explored for learning semantic parsers is asking users to validate the execution results (Clarke et al., 2010; Artzi and Zettlemoyer, 2013; Iyer et al., 2017). While appealing, in practice it may be a difficult task for real users because they would not need to ask the question if they knew the answer in the first place. We instead aim to learn semantic parsers from fine-grained interaction where users only need to answer simple questions covered by their background knowledge (Figure 1). However, learning signals from such fine-grained interactions are bound to be sparse because the system needs to avoid asking too many questions and overwhelming the user, which poses a challenge for learning.

To tackle the problem, we propose NEIL, a novel *aNnotation-Efficient Imitation Learning* algorithm for learning semantic parsers from such sparse, fine-grained demonstrations: The agent (semantic parser) only requests for demonstrations when it is uncertain about a state (parsing step). For certain/confident states, actions chosen by the current policy are deemed correct and are executed to continue parsing. The policy is updated iteratively in a Dataset Aggregation fashion (Ross et al., 2011). In each iteration, all the state-action pairs, demonstrated or confident, are included to form a new training set and train a new policy in a supervised way. Intuitively, using confident state-action pairs for training mitigates the sparsity issue, but it may also introduce training bias. We provide a theoretical analysis and show that, under mild assumptions, the impact of the bias and the quality of the NEIL policy can be controlled by tuning the policy initialization and confidence estimation accuracy.

We also empirically compare NEIL with a number of baselines on the text-to-SQL parsing task. On the WikiSQL (Zhong et al., 2017) dataset, we show that, *when bootstrapped using only 10% of the training data,* NEIL *can achieve almost the same test accuracy (2% absolute loss) as the full expert annotation baseline, while requiring less than 10% of the annotations that the latter needs*, without even taking into account the different unit cost of annotations from users vs. domain experts. We also show that the quality of the final policy is largely determined by the quality of the initial policy, which provides empirical support for the theoretical analysis. Finally, we demonstrate that NEIL can generalize to more complex semantic parsing tasks such as Spider (Yu et al., 2018).

2 Related Work

Interactive Semantic Parsing. Our work extends the recent idea of leveraging system-user interaction to improve semantic parsing on the fly (Li and Jagadish, 2014; He et al., 2016; Chaurasia and Mooney, 2017; Su et al., 2018; Gur et al., 2018; Yao et al., 2019a,b; Elgohary et al., 2020; Zeng et al., 2020; Semantic Machines et al., 2020). Gur et al. (2018) built a neural model to identify and correct error spans in generated queries via dialogues. Yao et al. (2019b) formalized a model-based intelligent agent MISP, which enables user interaction via a policy probability-based uncertainty estimator, a grammar-based natural language generator, and a multi-choice question-answer interaction design. More recently, Elgohary et al. (2020) crowdsourced a dataset for fixing incorrect SQL queries using free-form natural language feedback. Semantic Machines et al. (2020) constructed a contextual semantic parsing dataset where agents could trigger conversations to handle exceptions such as ambigu-

ous or incomplete user commands. In this work, we seek to continually improve semantic parsers from such user interaction, a topic that is not carefully studied by the aforementioned work.

Interactive Learning from Feedback. Learning interactively from user feedback has been studied in many NLP tasks (Sokolov et al., 2016; Wang et al., 2016, 2017; Nguyen et al., 2017; Gao et al., 2018; Abujabal et al., 2018; Hancock et al., 2019; Kreutzer and Riezler, 2019). Most relevant to us, Hancock et al. (2019) constructed a self-feeding chatbot that improves itself from user satisfied responses and their feedback on unsatisfied ones.

In the field of semantic parsing, Clarke et al. (2010); Artzi and Zettlemoyer (2013); Iyer et al. (2017) learned semantic parsers from binary user feedback on whether executing the generated query yields correct results. However, often times (especially in information-seeking scenarios) it may not be very practical to expect end users able to validate the denotation correctness (e.g., consider validating an execution result "*103*" for the question "*how many students have a GPA higher than 3.5*" from a massive table). Active learning is also leveraged to save human annotations (Duong et al., 2018; Ni et al., 2020). Our work is complementary to this line of research as we focus on learning interactively from end users (not "teachers").

Imitation Learning. Traditional imitation learning algorithms (Daumé et al., 2009; Ross and Bagnell, 2010; Ross et al., 2011; Ross and Bagnell, 2014) iteratively execute and train a policy by collecting expert demonstrations for every policy decision. Despite its efficacy, the learning demands costly annotations from experts. In contrast, we save expert effort by selectively requesting demonstrations. This idea is related to active imitation learning (Chernova and Veloso, 2009; Kim and Pineau, 2013; Judah et al., 2014; Zhang and Cho, 2017). For example, Judah et al. (2014) assumed a "teacher" and actively requested demonstrations for most informative trajectories in the unlabeled data pool. Similar to us, Chernova and Veloso (2009) solicited demonstrations only for uncertain states. However, their algorithm simply abandons policy actions that are confident, leading to sparse training data. Instead, our algorithm utilizes confident policy actions to combat the sparsity issue and is additionally provided with a theoretical analysis.

Concurrent with our work, Brantley et al. (2020) studied active imitation learning for structured prediction tasks such as named entity recognition. Our work instead focuses on semantic parsing, which presents a unique challenge of *integrality*, i.e., the output sequence (a semantic parse) could only be correct *as a whole* (as opposed to *partially* correct) in order to yield the correct denotation. We therefore propose a new cost function (Section 5) to theoretically analyze the factors that affect the efficacy of learning semantic parsers via imitation.

3 Preliminaries

Formally, we assume the semantic parsing model generates a semantic parse by executing a sequence of actions a_t (parsing decisions) at each time step t. In practice, the definition of an action depends on the specific semantic parsing model, as we will illustrate shortly. A state s_t is then defined as a tuple of $(q, a_{1:t-1})$, where q is the initial natural language question and $a_{1:t-1} = (a_1, ..., a_{t-1})$ is the current partial parse. In particular, the initial state $s_1 = (q, \phi)$ contains only the question. Denote a semantic parser as $\hat{\pi}$, which is a *policy function* (Sutton and Barto, 2018) that takes a state s_t as input and outputs a probability distribution over the action space. The semantic parsing process can be formulated as sampling a *trajectory* τ by alternately observing a state and sampling an action from the policy, i.e., $\tau = (s_1, a_1 \sim \hat{\pi}(s_1), ..., s_T, a_T \sim \hat{\pi}(s_T))$, assuming a trajectory length T. The probability of the generated semantic parse becomes: $p_{\hat{\pi}}(a_{1:T}|s_1) = \prod_{t=1}^{T} p_{\hat{\pi}}(a_t|s_t)$.

An interactive semantic parser typically follows the aforementioned definition and requests the user's validation of a specific action a_t. Based on the feedback, a correct action a_t^* can be inferred to replace the original a_t. The parsing process continues with a_t^* afterwards.

In this work, we adopt MISP (Yao et al., 2019b) as the back-end interactive semantic parsing framework, given that it is a principled framework for this purpose and can generalize to various kinds of semantic parsers and logical forms. However, we note that our proposed algorithm is not limited to MISP; it instead depicts a general algorithm for learning semantic parsers from user interaction. We illustrate the application of MISP to a sketch-based parser, SQLova (Hwang et al., 2019), as follows. More details and another example of how it applies to a non-sketch-based parser EditSQL (Zhang et al., 2019) can be found in Appendix B.1.

Example. Consider the SQLova parser, which

generates a query by filling "slots" in a predefined SQL sketch "`SELECT Agg SCol WHERE WCol OP VAL`". To complete the SQL query in Figure 1, it first takes three steps: `SCol`="`School/Club Team`" (a_1), `Agg`="`COUNT`" (a_2) and `WCol`="`School/Club Team`" (a_3). MISP detects that a_3 is uncertain because its probability is lower than a pre-specified threshold. It validates a_3 with the user and corrects it with `WCol`="`Player`" (a_3^*). The parsing continues with `OP`="`=`" (a_4) and `VAL`="`jalen rose`" (a_5). Here, the trajectory length $T = 5$.

4 Learning Semantic Parsers from User Interaction

In this section, we present NEIL, an *aNnotation-Efficient Imitation Learning* algorithm that trains a parser from user interaction, without requiring a large amount of user feedback (or "annotations"). This property is particularly important for end user-facing systems in practical use. Note that while we apply NEIL to semantic parsing in this work, in principle it can also be applied to other structured prediction tasks (e.g., machine translation).

4.1 An Imitation Learning Formulation

Under the interactive semantic parsing framework, a learning algorithm intuitively can aggregate (s_t, a_t^*) pairs collected from user interactions and trains the parser to enforce a_t^* under the state $s_t = (q, a_{1:t-1})$. However, this is not achievable by conventional supervised learning since the training needs to be conducted in an *interactive* environment, where the partial parse $a_{1:t-1}$ is generated by the parser itself.

Instead, we formulate it as an imitation learning problem (Daumé et al., 2009; Ross and Bagnell, 2010). Consider the user as a *demonstrator*, then the derived action a_t^* can be viewed as an *expert demonstration* which is interactively sampled from the demonstrator's policy (or *expert policy*) π^*,[2] i.e., $a_t^* \sim \pi^*(s_t)$. The goal of our algorithm is thus to train policy $\hat{\pi}$ to imitate the expert policy π^*. A general procedure is described in Algorithm 1 (Line 1–9), where $\hat{\pi}$ is learned iteratively for every m user questions. In each iteration, the policy is retrained on an aggregated training data over the past iterations, following the Dataset Aggregation fashion in (Ross et al., 2011).

[2]We follow the imitation learning literature and use "expert" to refer to the imitation target, but the user in our setting by no means needs to be a "domain (SQL) expert".

Algorithm 1 The NEIL Algorithm

Input: Initial training data D_0, policy confidence threshold μ.
Output: A trained policy $\hat{\pi}$.

1: Initialize $D \leftarrow D_0$.
2: Initialize $\hat{\pi}_1$ by training it on D_0.
3: **for** $i = 1$ to N **do**
4: Observe m user questions $q_j, j \in [1, m]$;
5: $D_i \leftarrow \bigcup_{j=1}^{m}$ PARSE&COLLECT$(\mu, q_j, \hat{\pi}_i, \pi^*)$;
6: Aggregate dataset $D \leftarrow D \bigcup D_i$;
7: Train policy $\hat{\pi}_{i+1}$ on D using Eq. (1).
8: **end for**
9: **return** best $\hat{\pi}_i$ on validation.
10: **function** PARSE&COLLECT$(\mu, q, \hat{\pi}_i, \pi^*)$
11: Initialize $D_i' \leftarrow \emptyset$, $s_1 = (q, \phi)$.
12: **for** $t = 1$ to T **do**
13: Preview action $a_t = \arg\max_a \hat{\pi}_i(s_t)$;
14: **if** $p_{\hat{\pi}_i}(a_t|s_t) \geq \mu$ **then**
15: $w_t \leftarrow 1$;
16: Collect $D_i' \leftarrow D_i' \bigcup \{(s_t, a_t, w_t)\}$;
17: Execute a_t;
18: **else**
19: Trigger user interaction and derive expert demonstration $a_t^* \sim \pi^*(s_t)$;
20: $w_t \leftarrow 1$ if a_t^* is valid; 0 otherwise;
21: Collect $D_i' \leftarrow D_i' \bigcup \{(s_t, a_t^*, w_t)\}$;
22: Execute a_t^*.
23: **end if**
24: **end for**
25: **return** D_i'.
26: **end function**

4.2 Annotation-efficient Imitation Learning

Consider parsing a user question and collecting training data using the parser $\hat{\pi}_i$ in the i-th iteration (Line 5). A standard imitation learning algorithm such as DAGGER (Ross et al., 2011) usually requests expert demonstration a_t^* for every state s_t in the sampled trajectory. However, it requires a considerable amount of user annotations, which may not be practical when interacting with end users.

Instead, we propose to adopt an *annotation-efficient* learning strategy in NEIL, which saves user annotations by *selectively* requesting user interactions, as indicated in function PARSE&COLLECT. In each parsing step, the system first previews whether it is confident about its own decision a_t (Line 13–14), which is determined when its probability is no less than a threshold, i.e., $p_{\hat{\pi}_i}(a_t|s_t) \geq$

μ.[3] In this case, the algorithm executes and collects its own action a_t (Line 15–17); otherwise, a system-user interaction will be triggered and the derived demonstration $a_t^* \sim \pi^*(s_t)$ will be collected and executed to continue parsing (Line 19–22).

Denote a collected state-action pair as $(s_t, \tilde{a}_t)$, where $\tilde{a}_t$ could be a_t or a_t^* depending on whether an interaction is requested. To train $\hat{\pi}_{i+1}$ (Line 7), our algorithm adopts a *reduction-based* approach similar to DAGGER and reduces imitation learning to iterative supervised learning. Formally, we define our training loss function as a *weighted* negative log-likelihood:

$$\mathcal{L}(\hat{\pi}_{i+1}) = -\frac{1}{|D|} \sum_{(s_t, \tilde{a}_t, w_t) \in D} w_t \log p_{\hat{\pi}_i}(\tilde{a}_t | s_t), \quad (1)$$

where D is the aggregated training data over i iterations and w_t denotes the weight of $(s_t, \tilde{a}_t)$.

We consider assigning weight w_t in three cases: (1) For confident actions a_t, we set $w_t = 1$. This essentially treats the system's own confident actions as gold decisions, which resembles self-training (Scudder, 1965; Nigam and Ghani, 2000; McClosky et al., 2006). (2) For user-confirmed decisions (*valid demonstrations* a_t^*), such as enforcing a `WHERE` condition on "`Player`" in Figure 1, w_t is also set to 1 to encourage the parser to imitate the correct decisions from users. (3) For uncertain actions that cannot be addressed via human interactions (*invalid demonstrations* a_t^*, which are identified when the user selects "None of the above options" in Figure 1), we assign $w_t = 0$. This could happen when some of the incorrect precedent actions are not fixed. For example, in Figure 1, if the system missed correcting the `WHERE` condition on "`School/Club Team`", then whatever value it generates after "`WHERE School/Club Team=`" is wrong, and thus any action a_t^* derived from human feedback would be invalid. In this case, the system selects the next available option without further validation and continues parsing.

A possible training strategy to handle case (3) may set w_t to be negative, similar to Welleck et al. (2020). However, empirically we find this strategy fails to train the parser to correct its mistake in generating "`School/Club Team`" but rather disturbs the model training. By setting $w_t = 0$, the impact of unaddressed actions is removed from training. A similar solution is also adopted in Petrushkov et al. (2018); Kreutzer and Riezler (2019). As shown in Section 6, this way of training weight assignment enables stable improvement in iterative model learning while requiring fewer user annotations.

5 Theoretical Analysis

While NEIL enjoys the benefit of learning from a small amount of user feedback, one crucial question is whether it can still achieve the same level of performance as the traditional supervised approach (which trains a policy on full expert annotations, if one could afford that and manage the privacy risk). In this section, we prove that the performance gap between the two approaches is mainly determined by the learning policy's probability of trusting a confident action that turns out to be wrong, which can be controlled in practice.

Our analysis follows prior work (Ross and Bagnell, 2010; Ross et al., 2011) to assume a unified trajectory length T and an infinite number of training samples in each iteration (i.e., $m = \infty$ in Algorithm 1), such that the state space can be full explored by the learning policy. An analysis under the "finite sample" case can be found in Appendix A.5.

5.1 Cost Function for Analysis

Unlike typical imitation learning tasks (e.g., Super Tux Kart (Ross et al., 2011)), in semantic parsing, there exists only one gold trajectory semantically identical to the question.[4] Whenever a policy action is different from the gold one, the whole trajectory will not yield the correct semantic meaning and the parsing is deemed failed. In other words, a well-performing semantic parser should be able to keep staying in the correct trajectory during the parsing. Therefore, for theoretical analysis, we only analyze a policy's performance when it is conditioned on a *gold partial parse*, i.e., $s_t \in d_{\pi^*}^t$, where $d_{\pi^*}^t$ is the state distribution in step t when executing the expert policy π^* for first t-1 steps. Let $\ell(s, \hat{\pi}) = 1 - p_{\hat{\pi}}(a = a^*|s)$ be the loss of $\hat{\pi}$ making a mistake at state s. We define the *cost* (i.e., the inverse *test-time* quality) of a policy as:

$$J(\hat{\pi}) = T\mathbb{E}_{s \sim d_{\pi^*}} \big[\ell(s, \hat{\pi})\big], \quad (2)$$

where $d_{\pi^*} = \frac{1}{T} \sum_{t=1}^{T} d_{\pi^*}^t$ denotes the average expert state distribution (assuming time step t is a

[3] The metric is shown effective for interactive semantic parsing in Yao et al. (2019b). Other confidence measures can also be explored, as we will discuss in Section 7.

[4] We assume a canonical order for swappable components in a parse. In practice, it may be possible, though rare, for one question to have multiple gold parses.

random variable uniformly sampled from $1 \sim T$). A detailed derivation is shown in Appendix A.1.

The *better* a policy $\hat{\pi}$ is, the *smaller* this cost becomes. Note that, by defining Eq. (2), we simplify the analysis from evaluating *the whole trajectory sampled from* $\hat{\pi}$ (as we do in experiments) to evaluating *the expected single-step loss of* $\hat{\pi}$ *conditioned on a gold partial parse*. This cost function makes the analysis easier and meanwhile reflects a consistent relative performance among algorithms for comparison. Next, we compare our NEIL algorithm with the supervised approach by analyzing the upper bounds of their costs.

5.2 Cost Bound of Supervised Approach

A fully supervised system trains a parser on expert-annotated $(q, a^*_{1:T})$ pairs, where the gold semantic parse $a^*_{1:T}$ can be viewed as generated by executing the expert policy π^*. This gives the policy $\hat{\pi}_{sup}$:

$$\hat{\pi}_{sup} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi^*}}[l(s, \pi)],$$

where Π is the policy space induced by the model architecture. A detailed derivation in Appendix A.2 shows the cost bound of the supervised approach:

Theorem 5.1. *For supervised approach, let* $\epsilon_N = \min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi^*}}[l(s, \pi)]$, *then* $J(\hat{\pi}_{sup}) = T\epsilon_N$.

The theorem gives an exact bound (as shown by the equality) since the supervised approach, given the "infinite sample" assumption, trains a policy under the same state distribution d_{π^*} as the one being evaluated in the cost function (Eq. (2)).

5.3 Cost Bound of NEIL Algorithm

Recall that, in each training iteration, NEIL samples trajectories by executing actions from both the previously learned policy $\hat{\pi}_i$ and the expert policy π^* (when an interaction is requested). Let π_i denote such a "mixture" policy. We derive the following cost bound of a NEIL policy $\hat{\pi}$:

$$J(\hat{\pi}) \leq \frac{T}{N} \sum_{i=1}^{N} \left[\mathbb{E}_{s \sim d_{\pi_i}}[\ell(s, \hat{\pi}_i)] + \ell_{max} ||d_{\pi_i} - d_{\pi^*}||_1 \right].$$

The bound is determined by two terms. The first term $\mathbb{E}_{s \sim d_{\pi_i}}[\ell(s, \hat{\pi}_i)]$ calculates the expected training loss of $\hat{\pi}_i$. Notice that, while the policy is *trained* on states induced by the mixture policy ($s \sim d_{\pi_i}$), what matters to its *test-time* quality is the policy's performance conditioned on a gold partial parse ($s \sim d_{\pi^*}$ in Eq. (2)). This state discrepancy, which does not exist in the supervised approach, explains the performance loss of NEIL, and is bounded by the second term $\ell_{max} ||d_{\pi_i} - d_{\pi^*}||_1$, the weighted L_1 distance between d_{π_i} and d_{π^*}. To bound the two terms, we employ a "no-regret" assumption (Kakade and Tewari, 2009; Ross et al., 2011, see Appendix A.3–A.4 for details), which gives the theorem:

Theorem 5.2. *For the proposed* NEIL *algorithm, if* N *is* $\tilde{O}(T)$, *there exists a policy* $\hat{\pi} \in \hat{\pi}_{1:N}$ *s.t.* $J(\hat{\pi}) \leq T\left[\epsilon_N + \frac{2T\ell_{max}}{N} \sum_{i=1}^{N} e_i\right] + O(1)$.

Here, $\epsilon_N = \min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{s \sim d_{\pi_i}}[\ell(s, \pi)]$ denotes the best expected policy loss in *hindsight*, and e_i denotes the probability that $\hat{\pi}_i$ does not query the expert policy (i.e., being confident) but its own action is wrong under d_{π^*}.

We note that a no-regret algorithm requires convexity of the loss function (Hazan et al., 2007; Kakade and Tewari, 2009), which is not satisfied by neural network-based semantic parsers. In general, proving theorems under a non-convex case is not trivial. Therefore, we follow the common practice (e.g., Kingma and Ba (2015); Reddi et al. (2018)) to theoretically analyze the convex case while empirically demonstrating the performance of our NEIL algorithm with non-convex loss functions (i.e., when it applies to neural semantic parsers). More accurate regret bound for non-convex cases will be studied in the future.

Remarks. Compared with the supervised approach (Theorem 5.1), NEIL's cost bound additionally contains a term of $\frac{1}{N} \sum_{i=1}^{N} e_i$, which, as we expect, comes from the aforementioned state discrepancy. Intuitively, if a learning policy frequently executes its own but wrong actions in training, the resulting training states d_{π_i} will greatly deviate from the gold ones d_{π^*}.

This finding inspires us to restrict the performance gap by reducing the learning policy's error rate *when it does not query the expert*. Empirically this can be achieved by: (1) *Accurate confidence estimation*, so that actions deemed confident are generally correct, and (2) *Moderate policy initialization*, such that in general the policy is less likely to make wrong actions throughout the iterative training. For (1), we set a high confidence threshold μ=0.95, which is demonstrated reliable for MISP (Yao et al., 2019b). We then empirically validate (2) in experiments.

6 Experiments

In this section, we conduct experiments to demonstrate the annotation efficiency of our NEIL algo-

rithm and that it can train semantic parsers for high performance when the parsers are reasonably initialized, which verifies our theoretical analysis.

6.1 Experimental Setup

We compare various systems on the WikiSQL dataset (Zhong et al., 2017). The dataset contains large-scale annotated question-SQL pairs (56,355 pairs for training) and thus serves as a good resource for experimenting iterative learning. For the base semantic parser, we choose SQLova (Hwang et al., 2019), one of the top-performing models on WikiSQL, to ensure a reasonable model capacity in terms of data utility along iterative training.

We experiment each system with three parser initialization settings, using 10%, 5% and 1% of the total training data. During iterative learning, questions from the remaining training data arrive in a random order to simulate user questions and we simulate user feedback by directly comparing the synthesized query with the gold one. In each iteration, all systems access exactly the same user questions. Depending on how they solicit feedback, each system collects a different number of annotations. At the end of each iteration, we update each system by retraining its parser on its accumulated annotations and the initial training data, and report its *(exact) query match accuracy* on the test set. We also report the *accumulated number of annotations* that each system has requested after each training iteration, in order to compare their annotation efficiency.

In experiments, we consider every 1,000 user questions as one training iteration (i.e., m=1,000 in Algorithm 1). We repeat the whole iterative training for three runs and report average results. *Reproducible details* are included in Appendix B.

6.2 System Comparison

We denote our system as **MISP-NEIL** since it leverages MISP in the back end of NEIL. We compare it with the traditional supervised approach (denoted as **Full Expert**). To investigate the skyline capability of our system, we also present a variant called **MISP-NEIL***, which is *assumed* with perfect confidence measurement and interaction design, so that it can precisely identify and correct its mistakes during parsing. This is implemented by allowing the system to compare its synthesized query with the gold one. Note that this is not a realized automatic system; we show its performance as an upper bound of MISP-NEIL.

On the other hand, although execution feedback-based learning systems (Clarke et al., 2010; Artzi and Zettlemoyer, 2013; Iyer et al., 2017) may not be very practical for end users, we include them nonetheless in the interest of a comprehensive comparison. This leads to two baselines. The **Binary User** system requests binary user feedback on whether *executing* the generated SQL query returns correct database results and collects only queries with correct results to further improve the parser. The **Binary User+Expert** system additionally collects full expert SQL annotations when the generated SQL queries do not yield correct answers.

Given the completely different nature of annotations from Binary User (which validate the *denotation*) and those from Full Expert and MISP-NEIL (which validate a semantic parse's *constituents*), there may not exist a universally fair way to convert one's annotation consumption into the other's. Therefore, in the following sections, we only present and discuss Binary User(+Expert) in terms of their parsing accuracy under different training iterations. To give an estimation of their annotation efficiency for reference, we design a compromised annotation calculation metric for Binary User(+Expert) and include their results on WikiSQL validation set in Appendix C.

Finally, while our MISP-NEIL and the aforementioned baselines all leverage feedback from users or domain experts, an interesting question is how much gain they could obtain compared with using no annotation or feedback at all. To this end, we compare the systems with a **Self Train** baseline (Scudder, 1965; Nigam and Ghani, 2000; McClosky et al., 2006). In each iteration, this baseline collects SQL queries generated *by itself* as the new gold annotations for further training. We additionally apply a confidence threshold to improve the collection quality, i.e., only SQL queries with probability $p_{\tilde{\pi}}(a_{1:T}|s_1)$ greater than 0.5 are included. This strategy empirically leads to better performance. Intuitively, we expect Self Train to perform no better than any other systems in our experiments, since no human feedback is provided to correct mistakes in its collection.

6.3 Experimental Results

We evaluate each system by answering two research questions (RQs):

- *RQ1: Can the system improve a parser without requiring a large amount of annotations?*

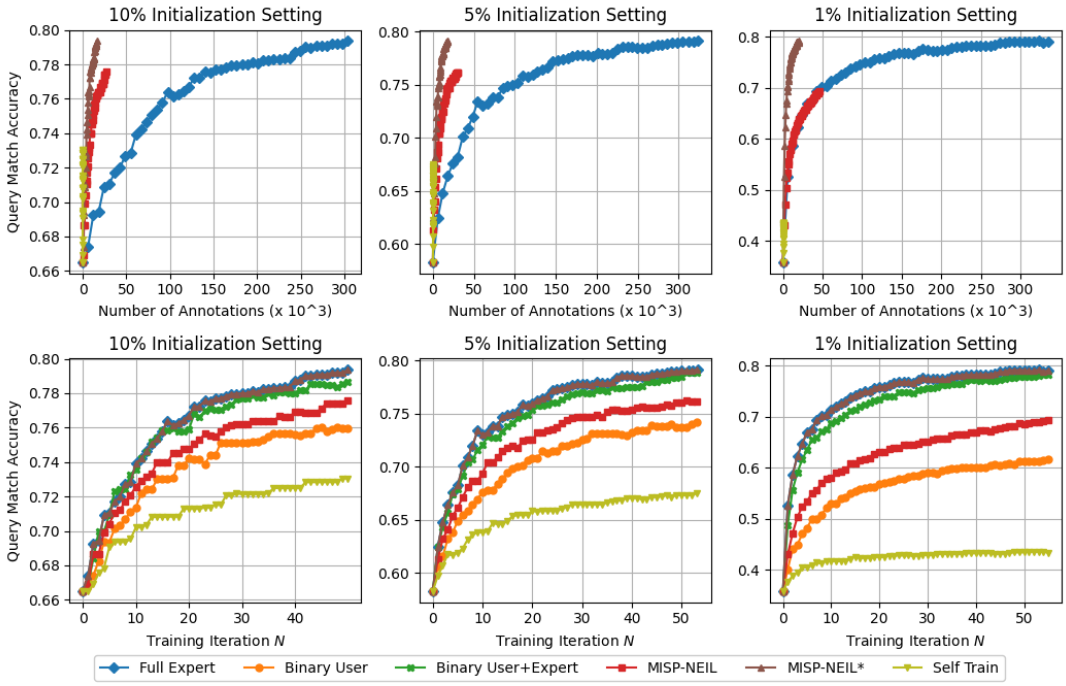


Figure 2: Parsing accuracy on WikiSQL test set when systems are trained with various numbers of user/expert annotations (top) and for different iterations (bottom). We experiment with three initialization settings, using 10%, 5% and 1% of the training data respectively. Results on validation set can be found in Appendix C.

- *RQ2: For interactive systems, while requiring weaker supervision, can they train the parser to reach a performance comparable to the traditional supervised system?*

For *RQ1*, we measure the number of *user/expert* annotations that a system requires to train a parser. For Full Expert, this number equals the trajectory length of the gold query (e.g., 5 for the query in Figure 1); for MISP-NEIL and MISP-NEIL*, it is the number of user interactions during training. Note that while we do not differentiate the actual (e.g., time/financial) cost of users from that of experts in this aspect, we emphasize that our system enjoys an additional benefit of collecting training examples from a much cheaper and more abundant source. For Self Train, the number of annotations is always zero since it does not request any human feedback for the online user questions.

Our results in Figure 2 (top) demonstrate that MISP-NEIL consistently consumes a comparable or smaller amount of annotations to train the parser to reach the same parsing accuracy. Figure 5 in Appendix further shows that, on average, it requires no more than one interaction for each user question along the training. Particularly in the 10% initialization setting, MISP-NEIL uses less than 10% of the total annotations that Full Expert needs in the end. Given the limited size of WikiSQL training set, the simulation experiments currently can only show MISP-NEIL's performance under a small number of annotations. However, we expect this gain to continue as it receives more user questions in the long-term deployment.

To answer *RQ2*, Figure 2 (bottom) compares each system's accuracy after they have been trained for the same number of iterations. The results demonstrate that when a semantic parser is moderately initialized (10%/5% initialization setting), MISP-NEIL can further improve it to reach a comparable accuracy as Full Expert (0.776/0.761 vs. 0.794 in the last iteration). In the extremely weak 1% initialization setting (using only around 500 initial training examples), all interactive learning systems suffer from a huge performance loss. This is consistent with our finding in theoretical analysis (Section 5). In Appendix C.2, we plot the value of e_i, the probability that $\hat{\pi}_i$ makes a confident but wrong decision given a gold partial parse, showing that a better initialized policy generally obtains a smaller e_i throughout the training and thus a tighter cost bound.

Our system also surpasses Binary User. We find

that the inferior performance of Binary User is mainly due to the "spurious program" issue (Guu et al., 2017), i.e., a SQL query having correct execution results can still be incorrect in terms of semantics. MISP-NEIL circumvents this issue by directly validating the *semantic meaning* of intermediate parsing decisions. The performance of Binary User+Expert is close to Full Expert as it has additionally involved expert annotations on a considerable number of user questions, which on the other hand also leads to extra annotation overhead.

When it is assumed with perfect interaction design and confidence estimator, MISP-NEIL* shows striking superiority in both aspects. Since it always corrects wrong decisions immediately, MISP-NEIL* can collect and derive the same training examples as Full Expert, and thus trains the parser to Full Expert's performance level in Figure 2 (bottom). However, it requires only 6% of the annotations that Full Expert needs (Figure 2, top). These observations imply large room for MISP-NEIL to be improved in the future.

Finally, we observe that all feedback-based learning systems outperform Self Train dramatically (Figure 2, bottom). This verifies the benefit of learning from human feedback.

6.4 Generalize to Complex SQL Queries

We next investigate whether MISP-NEIL can generalize to the complex SQL queries in the Spider dataset (Yu et al., 2018), which can contain complicated keywords like `GROUP BY`. For the base semantic parser, we choose EditSQL (Zhang et al., 2019), one of the open-sourced top models on Spider. Given the small size of Spider (7,377 question-SQL pairs for training after data cleaning; see Appendix B.3 for details), we only experiment with one initialization setting, using 10% of the training set. Since EditSQL does not predict the specific values in a SQL query (e.g., "`jalen rose`" in Figure 1), we cannot execute the generated query to simulate the binary execution feedback. Therefore, we only compare our system with Full Expert and Self Train. Parsers are evaluated on Spider Dev set since its test set is not publicly available.

Figure 3 (top) shows that MISP-NEIL and MISP-NEIL* consistently achieve comparable or better annotation efficiency while enjoying the advantage of learning from end user interaction. We expect this superiority to continue as the systems receive more user questions beyond Spider. Meanwhile,

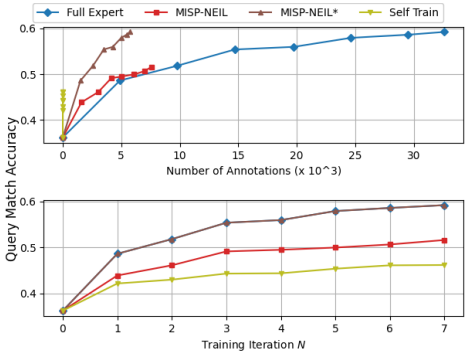


Figure 3: Parsing accuracy on Spider Dev set when systems are trained with various numbers of user/expert annotations and for different iterations.

we also notice that the gain is smaller and MISP-NEIL suffers from a large performance loss compared with Full Expert (Figure 3, bottom), due to the poor parser initialization and the SQL query complexity. This can be addressed via adopting better interaction designs and more accurate confidence estimation, as shown by MISP-NEIL*. Similarly as in WikiSQL experiments, Self Train performs worse than human-in-the-loop learning systems, as there is no means to correct wrong predictions in its collected annotations.

7 Conclusion and Future Work

Our work shows the possibility of continually learning semantic parsers from fine-grained end user interaction. As a pilot study, we experiment systems with simulated user interaction. One important future work is thus to conduct large-scale user studies and train parsers from real user interaction. This is not trivial and has to account for uncertainties such as noisy user feedback. We also plan to derive a more realistic formulation of user/expert annotation costs by analyzing real user statistics (e.g., average time spent on each question).

In experiments, we observe that neural semantic parsers tend to be overconfident and training them with more data does not mitigate this issue. In the future, we will look into more accurate confidence measure via neural network calibration (Guo et al., 2017) or using machine learning components (e.g., answer triggering (Zhao et al., 2017) or a reinforced active selector (Fang et al., 2017)).

Finally, we believe our algorithm can be applied to save annotation effort for other NLP tasks, especially the low-resource ones (Mayhew et al., 2019).

Acknowledgments

We would like to thank Khanh Nguyen and the anonymous reviewers for their helpful discussions or comments. The research conducted by the Ohio State University authors was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, NSF CAREER #1942980, Fujitsu gift grant, and Ohio Supercomputer Center (Center, 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein. The Spider dataset (Yu et al., 2018) is distributed under the CC BY-SA 4.0 license.

References

Abdalghani Abujabal, Rishiraj Saha Roy, Mohamed Yahya, and Gerhard Weikum. 2018. Never-ending learning for open-domain question answering over knowledge bases. In *Proceedings of the 2018 World Wide Web Conference*, pages 1053–1062.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Kazuoki Azuma. 1967. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal, Second Series*, 19(3):357–367.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Kianté Brantley, Amr Sharaf, and Hal Daumé, III. 2020. Active imitation learing with noisy guidance. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*.

Ohio Supercomputer Center. 1987. Ohio supercomputer center. http://osc.edu/ark:/19495/f5s1ph73.

Shobhit Chaurasia and Raymond J Mooney. 2017. Dialog for language to code. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 175–180.

Sonia Chernova and Manuela Veloso. 2009. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 18–27. Association for Computational Linguistics.

Hal Daumé, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.

Li Dong, Chris Quirk, and Mirella Lapata. 2018. Confidence modeling for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 743–753.

Long Duong, Hadi Afshar, Dominique Estival, Glen Pink, Philip R Cohen, and Mark Johnson. 2018. Active learning for deep semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 43–48.

Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your parser: Interactive text-to-SQL with natural language feedback. In *Annual Conference of the Association for Computational Linguistics (ACL 2020)*.

Meng Fang, Yuan Li, and Trevor Cohn. 2017. Learning how to active learn: A deep reinforcement learning approach. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 595–605.

Yang Gao, Christian M Meyer, and Iryna Gurevych. 2018. APRIL: Interactively learning to summarise by combining active preference learning and reinforcement learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4120–4130.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1339–1349.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062.

Braden Hancock, Antoine Bordes, Pierre-Emmanuel Mazare, and Jason Weston. 2019. Learning from dialogue after deployment: Feed yourself, chatbot! In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3667–3684.

Elad Hazan, Amit Agarwal, and Satyen Kale. 2007. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192.

Luheng He, Julian Michael, Mike Lewis, and Luke Zettlemoyer. 2016. Human-in-the-loop parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2337–2342.

Wassily Hoeffding. 1994. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on WikiSQL with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973.

Kshitij Judah, Alan P Fern, Thomas G Dietterich, and Prasad Tadepalli. 2014. Active imitation learning: Formal and practical reductions to iid learning. *Journal of Machine Learning Research*, 15:4105–4143.

Sham M Kakade and Ambuj Tewari. 2009. On the generalization ability of online strongly convex programming algorithms. In *Advances in Neural Information Processing Systems*, pages 801–808.

Beomjoon Kim and Joelle Pineau. 2013. Maximum mean discrepancy imitation learning. *Robotics: Science and systems*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Julia Kreutzer and Stefan Riezler. 2019. Self-regulated interactive sequence-to-sequence learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 303–315.

Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Natasha Lomas. 2019. Google ordered to halt human review of voice AI recordings over privacy risks. https://techcrunch.com/2019/08/02/google-ordered-to-halt-human-review-of-voice-ai-recordings-over-privacy-risks/. Accessed: 2020-04-28.

Stephen Mayhew, Snigdha Chaturvedi, Chen-Tse Tsai, and Dan Roth. 2019. Named entity recognition with partially annotated training data. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 645–655.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159.

Khanh Nguyen, Hal Daumé III, and Jordan Boyd-Graber. 2017. Reinforcement learning for bandit neural machine translation with simulated human feedback. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1464–1474.

Ansong Ni, Pengcheng Yin, and Graham Neubig. 2020. Merging weak and active supervision for semantic parsing. In *Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, New York, USA.

Kamal Nigam and Rayid Ghani. 2000. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 86–93.

Pavel Petrushkov, Shahram Khadivi, and Evgeny Matusov. 2018. Learning from chunk-based feedback in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 326–331.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. 2018. On the convergence of adam and beyond. In *International Conference on Learning Representations*.

Stéphane Ross and Drew Bagnell. 2010. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668.

Stéphane Ross and J. Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *ArXiv*, abs/1406.5979.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.

H Scudder. 1965. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371.

Semantic Machines, Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Artem Sokolov, Julia Kreutzer, Christopher Lo, and Stefan Riezler. 2016. Learning structured predictors from bandit feedback for interactive nlp. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1610–1620.

Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. 2017. Building natural language interfaces to web apis. In *Proceedings of the International Conference on Information and Knowledge Management*.

Yu Su, Ahmed Hassan Awadallah, Miaosen Wang, and Ryen W White. 2018. Natural language interfaces with fine-grained user interaction: A case study on web APIs. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Sida I Wang, Samuel Ginn, Percy Liang, and Christopher D Manning. 2017. Naturalizing a programming language via interactive learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 929–938.

Sida I Wang, Percy Liang, and Christopher D Manning. 2016. Learning language games through interaction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2368–2378.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2020. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*.

William A Woods. 1973. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the American Federation of Information Processing Societies Conference*.

Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, and Huan Sun. 2019a. Interactive semantic parsing for if-then recipes via hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2547–2554.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019b. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5450–5461.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

Jichuan Zeng, Xi Victoria Lin, Steven CH Hoi, Richard Socher, Caiming Xiong, Michael Lyu, and Irwin King. 2020. Photon: A robust cross-domain text-to-SQL system. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 204–214.

Luke S Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 658–666.

Jiakai Zhang and Kyunghyun Cho. 2017. Query-efficient imitation learning for end-to-end simulated driving. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5341–5352.

Jie Zhao, Yu Su, Ziyu Guan, and Huan Sun. 2017. An end-to-end deep framework for answer triggering with a novel group-level objective. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1276–1282.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Theoretical Analysis in Infinite Sample Case

In this section, we give a detailed theoretical analysis to derive the cost bounds of the supervised approach and our proposed NEIL algorithm (Section 4). Following Ross et al. (2011), we first focus the proof on an *infinite* sample case, which assumes an infinite number of samples to train a policy in each iteration (i.e., $m = \infty$ in Algorithm 1), such that the state space in training can be full explored by the learning policy.

As an overview, we start the analysis by introducing the "cost function" we use to analyze each policy in Appendix A.1, which represents an inverse quality of a policy. In Appendix A.2, we derive the bound of the cost of the supervised approach. Appendix A.3 and Appendix A.4 then discuss the cost bound of our proposed NEIL algorithm. Finally, in Appendix A.5, we show the cost bound of NEIL in *finite* sample case.

A.1 Cost Function for Analysis

In a semantic parsing task, whenever a policy action is different from the gold one, the whole trajectory cannot yield the correct semantic meaning and the parsing is deemed failed. Therefore, we analyze a policy's performance only when it is conditioned on *a gold partial parse*. Intuitively, a policy with better quality should have a higher parsing accuracy under a gold partial parse, so that it is more likely to sample a completely correct trajectory in inference time.

Given a question q and denoting $a^*_{1:t}$ as the gold partial trajectory sampled by the expert policy π^*, we first define the *cost* of sampling a partial trajectory $a_{1:t} = (a_1, ..., a_t)$ as:

$$C(q, a_{1:t}) = \begin{cases} 0 & \text{if } a_{1:t} = a^*_{1:t} \\ 1 & \text{otherwise} \end{cases}.$$

In other words, a sampled partial trajectory is correct if and only if it is the same as the gold partial parse. Based on this definition, we further define the expected cost of policy $\hat{\pi}$ *in a single time step* t (given the question q) as:

$$\begin{aligned} C^t_{\hat{\pi}}(q) &= \mathbb{E}_{a_{1:t-1}\sim\pi^*}\mathbb{E}_{a_t\sim\hat{\pi}}[C(q, a_{1:t})] \\ &= \mathbb{E}_{a_{1:t-1}\sim\pi^*}[1 - p_{\hat{\pi}}(a_t = a^*_t|q, a_{1:t-1})]. \end{aligned}$$

Here, $a_{1:t-1} \sim \pi^*$ denotes a gold partial parse till the (t-1)-th step, which is obtained by executing the expert policy π^* for the first t-1 steps (given q), and $p_{\hat{\pi}}(a_t = a^*_t|q, a_{1:t-1})$ denotes the probability that $\hat{\pi}$ samples action a^*_t given a question q and a partial parse $a_{1:t-1}$. By taking an expectation over all questions $q \in \mathcal{Q}$, we have the following derivations:

$$\begin{aligned} \mathbb{E}_{q\in\mathcal{Q}}[C^t_{\hat{\pi}}(q)] &= \mathbb{E}_{q\in\mathcal{Q}, a_{1:t-1}\sim\pi^*}[1 - p_{\hat{\pi}}(a_t = a^*_t|q, a_{1:t-1})] \\ &= \mathbb{E}_{s_t\sim d^t_{\pi^*}}[1 - p_{\hat{\pi}}(a_t = a^*_t|s_t)]. \end{aligned}$$

The second equality holds by the definition $s_t = (q, a_{1:t-1})$, and $d^t_{\pi^*}$ is the "expert state distribution" in step t when executing the expert policy π^* for first t-1 steps. In this analysis, we follow Ross and Bagnell (2010); Ross et al. (2011) to assume a unified decision length T. By summing up the above expected cost over the T steps, we define the *cost* (i.e., the inverse *test-time* quality) of policy $\hat{\pi}$:

$$\begin{aligned} J(\hat{\pi}) &= \sum_{t=1}^{T} \mathbb{E}_{q\in\mathcal{Q}}[C^t_{\hat{\pi}}(q)] \\ &= \sum_{t=1}^{T} \mathbb{E}_{s_t\sim d^t_{\pi^*}}[1 - p_{\hat{\pi}}(a_t = a^*_t|s_t)]. \end{aligned}$$

Denote $\ell(s, \hat{\pi}) = 1 - p_{\hat{\pi}}(a = a^*|s), a \sim \hat{\pi}(s), a^* \sim \pi^*(s)$ as the "single-step loss function", which is bounded within $[0, 1]$, then the cost of policy $\hat{\pi}$ can be simplified as:

$$\begin{aligned} J(\hat{\pi}) &= \sum_{t=1}^{T} \mathbb{E}_{s_t\sim d^t_{\pi^*}}\big[\ell(s_t, \hat{\pi})\big] \\ &= T\mathbb{E}_{t\sim\mathcal{U}(1,T)}\mathbb{E}_{s_t\sim d^t_{\pi^*}}\big[\ell(s_t, \hat{\pi})\big] \\ &= T\mathbb{E}_{s\sim d_{\pi^*}}\big[\ell(s, \hat{\pi})\big], \end{aligned} \tag{3}$$

where $d_{\pi^*} = \frac{1}{T}\sum_{t=1}^{T} d^t_{\pi^*}$ is the average expert state distribution, when we assume the time step t to be a random variable under the uniform distribution $\mathcal{U}(1, T)$ (the second equality).

A.2 Cost Bound of Supervised Approach

In this section, we analyze the cost bound of the supervised approach. Recall that the supervised approach trains a policy $\hat{\pi}$ using the standard supervised learning algorithm with supervision from π^* at every decision step. Therefore, it finds the best policy $\hat{\pi}_{sup}$ on infinite samples as:

$$\hat{\pi}_{sup} = \arg\min_{\pi\in\Pi} \mathbb{E}_{s\sim d_{\pi^*}}[\ell(s, \pi)], \tag{4}$$

where Π denotes the policy space induced by the model architecture, and the expectation over s is

sampled from the whole d_{π^*} state space because of the "infinite sample" assumption. The supervised approach thus obtains the following cost bound:

$$\begin{aligned} J(\hat{\pi}_{sup}) =& T\mathbb{E}_{s\sim d_{\pi^*}}[\ell(s,\hat{\pi}_{sup})] \\ =& T\min_{\pi\in\Pi}\mathbb{E}_{s\sim d_{\pi^*}}[\ell(s,\pi)]. \end{aligned}$$

This gives the following theorem:

Theorem A.1. *For supervised approach, let* $\epsilon_N = \min_{\pi\in\Pi}\mathbb{E}_{s\sim d_{\pi^*}}[\ell(s,\pi)]$*, then* $J(\hat{\pi}_{sup}) = T\epsilon_N$.

The cost bound of the supervised approach represents its exact performance as implied by the equality. This is because the approach trains a policy (Eq. (4)) under the same state distribution d_{π^*} (given the "infinite sample" assumption) as in evaluation (Eq. (3)). As we will show next, the proposed NEIL algorithm breaks this consistency while enjoying the benefit of high annotation efficiency, which explains the performance gap.

A.3 No-regret Assumption

Before showing the cost bound of our NEIL algorithm, we introduce a "no-regret" assumption (Kakade and Tewari, 2009; Ross et al., 2011) that is leveraged in the derivation.

Assumption A.1. *No-regret assumption. Define* $\ell_i(\pi) = \mathbb{E}_{s\sim d_{\pi_i}}[l(s,\pi)]$ *and* $\epsilon_N = \min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N}\ell_i(\pi)$*, then*

$$\frac{1}{N}\sum_{i=1}^{N}\ell_i(\hat{\pi}_i) - \epsilon_N \leq \gamma_N$$

for $\lim_{N\to\infty}\gamma_N = 0$ *(usually* $\gamma_N \in \tilde{O}(\frac{1}{N})$*).*

This assumption characterizes an important policy learning pattern: As a policy is trained for an infinite number of iterations, on average, its expected training loss ($\frac{1}{N}\sum_{i=1}^{N}\ell_i(\hat{\pi}_i)$) will converge to the loss of the best policy in hindsight (ϵ_N). In our scenario, this assumption implies that, while our policy is trained on online labels from both the expert policy π^* (when it is queried) and the previously learned policy $\hat{\pi}_i$ (when the agent is confident), it still gradually fits to the best policy over the same state space in training (d_{π_i}). In other words, the likely noisy labels from $\hat{\pi}_i$ do not harm the model fitting to the expert policy in general.

Many no-regret algorithms (Hazan et al., 2007; Kakade and Tewari, 2009) that guarantee $\gamma_N \in \tilde{O}(\frac{1}{N})$ require convexity or strong-convexity of the loss function. However, the loss function used in our application, which is built on the top of a deep neural network model, does not satisfy this requirement. In general, proving theorems under a non-convex case is not trivial. In this analysis, we follow the common practice (see Kingma and Ba (2015); Reddi et al. (2018) for example) to theoretically analyze the convex case while empirically demonstrating the non-convex case. A more accurate regret bound for non-convex neural networks (which may result in a slower γ_N convergence speed with respect to N) can be studied in the future.

A.4 Cost Bound of NEIL Algorithm

As shown in Algorithm 1, NEIL produces a sequence of policies $\hat{\pi}_{1:N} = (\hat{\pi}_1, \hat{\pi}_2, ..., \hat{\pi}_N)$, where N is the number of training iterations, and returns the one with the best test-time performance on validation set as $\hat{\pi}$. In training, the algorithm executes actions from both the learning policy $\hat{\pi}_i$ (when the model is confident) and the expert policy π^*. We denote this "mixture" policy as π_i. Then for the first N iterations, we have the cost bound of NEIL as:

$$\begin{aligned} J(\hat{\pi}) =& \min_{\hat{\pi}'\in\hat{\pi}_{1:N}} T\mathbb{E}_{s\sim d_{\pi^*}}\left[\ell(s,\hat{\pi}')\right] \\ \leq& \frac{T}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim d_{\pi^*}}\left[\ell(s,\hat{\pi}_i)\right] \\ \leq& \frac{T}{N}\sum_{i=1}^{N}\left[\mathbb{E}_{s\sim d_{\pi_i}}[\ell(s,\hat{\pi}_i)] + \ell_{max}||d_{\pi_i} - d_{\pi^*}||_1\right]. \end{aligned} \tag{5}$$

From the last inequality, we can see that the cost bound of NEIL is restricted by two terms. The first term $\mathbb{E}_{s\sim d_{\pi_i}}[\ell(s,\hat{\pi}_i)]$ denotes the expected loss of $\hat{\pi}_i$ under the states induced by π_i *during training* (under the "infinite sample" assumption, as mentioned in the beginning of the analysis). By applying the no-regret assumption (Assumption A.1), this term can be bound by $\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim d_{\pi_i}}[\ell(s,\hat{\pi}_i)] \leq \epsilon_N + \gamma_N$. Here, $\epsilon_N = \min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N}\ell_i(\pi)$ denotes the best expected training loss in hindsight.

The second term denotes the L_1 distance between state distributions induced by π_i and π_i^*, weighted by the maximum loss value l_{max} that $\hat{\pi}_i$ encounters over the training. As we notice, unlike the supervised approach, NEIL *trains* a policy under d_{π_i}, while what matters to its *test-time* quality is its performance on the state distribution d_{π^*} (Eq. (3)). This discrepancy explains the performance loss of our algorithm compared to the supervised approach and is bounded by the aforementioned L_1 distance. To further bound this term,

we define e_i as the probability that $\hat{\pi}_i$ makes a confident (i.e., without querying the expert policy) but wrong action under d_{π^*}, and introduce the following lemma:

Lemma A.1. $||d_{\pi_i} - d_{\pi^*}||_1 \leq 2Te_i$.

Proof. Let β_{it} be the probability of querying the expert policy under $d^t_{\pi^*}$, $\tilde{\epsilon}_{it}$ the error rate of $\hat{\pi}_i$ under $d^t_{\pi^*}$ (w.r.t. π^*), and d any state distribution besides d_{π^*}. We can then express d_{π_i} by:

$$d_{\pi_i} = \prod_{t=1}^{T}(\beta_{it} + (1-\beta_{it})(1-\tilde{\epsilon}_{it}))d_{\pi^*} + (1 - \prod_{t=1}^{T}(\beta_{it} + (1-\beta_{it})(1-\tilde{\epsilon}_{it})))d.$$

The distance between d_{π_i} and d_{π^*} thus becomes

$$\begin{aligned}
&||d_{\pi_i} - d_{\pi^*}||_1 \\
=&(1 - \prod_{t=1}^{T}(\beta_{it} + (1-\beta_{it})(1-\tilde{\epsilon}_{it})))||d - d_{\pi^*}||_1 \\
\leq& 2(1 - \prod_{t=1}^{T}(\beta_{it} + (1-\beta_{it})(1-\tilde{\epsilon}_{it}))) \\
\leq& 2\sum_{t=1}^{T}[1 - (\beta_{it} + (1-\beta_{it})(1-\tilde{\epsilon}_{it}))] \\
\leq& 2\sum_{t=1}^{T}[\tilde{\epsilon}_{it}(1-\beta_{it})] \\
\leq& 2\sum_{t=1}^{T} e_{it} \\
=& 2Te_i.
\end{aligned}$$

The second inequality uses $1 - \prod_{t=1}^{T} x_t \leq \sum_{t=1}^{T}(1-x_t)$, which holds when $x_t \in [0,1]$. □

By applying Assumption A.1 and Lemma A.1 to Eq. (3), we derive the following inequality:

$$J(\hat{\pi}) \leq T\big[\gamma_N + \epsilon_N + \frac{2T\ell_{max}}{N}\sum_{i=1}^{N} e_i\big].$$

Given a large enough N ($N \in \tilde{O}(T)$), by the no-regret assumption, we can further simplify the above as:

$$J(\hat{\pi}) \leq T\big[\epsilon_N + \frac{2T\ell_{max}}{N}\sum_{i=1}^{N} e_i\big] + O(1),$$

which leads to our theorem:

Theorem A.2. *For the proposed* NEIL *algorithm, if N is $\tilde{O}(T)$, there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ s.t. $J(\hat{\pi}) \leq T\big[\epsilon_N + \frac{2T\ell_{max}}{N}\sum_{i=1}^{N} e_i\big] + O(1)$.*

By comparing Theorem A.1 and Theorem A.2, it is obvious that the performance gap between NEIL and the supervised approach is bounded by the term around $\frac{1}{N}\sum_{i=1}^{N} e_i$. We discuss its implications in Section 5 and show that, in practice, this performance gap can be controlled by carefully initializing the policy and choosing a more accurate confidence estimator.

Discussion about MISP-NEIL*. In experiments, we consider a skyline instantiation of NEIL, called MISP-NEIL*. This instantiation is assumed with perfect confidence estimation and interaction design, such that it can precisely detect and correct its intermediate mistakes during parsing. Therefore, MISP-NEIL* presents an upper bound performance (i.e., the *tightest* cost bound) of NEIL. This can be interpreted theoretically. In fact, for MISP-NEIL*, e_i is always zero since the system has ensured that its policy action is correct when it does not query the expert policy. In this case, $d_{\pi_i} = d_{\pi^*}$, so $\epsilon_N = \min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim d_{\pi^*}}[l(s,\pi)] = \min_{\pi\in\Pi}\mathbb{E}_{s\sim d_{\pi^*}}[l(s,\pi)]$. Therefore, according to Theorem A.2, MISP-NEIL* has a cost bound of:

$$J(\hat{\pi}) \leq T\epsilon_N + O(1),$$

where $\epsilon_N = \min_{\pi\in\Pi}\mathbb{E}_{s\sim d_{\pi^*}}[l(s,\pi)]$.

By comparing this bound with the cost bound in Theorem A.1, it is observed that MISP-NEIL* shares the same cost bound as the supervised approach (except for the inequality relation and the constant). This is explainable since MISP-NEIL* indeed collects exactly the same training labels as the supervised approach.

A.5 Cost Bound of NEIL Algorithm in Finite Sample Case

The theorem in the previous section holds when the algorithm observes infinite trajectories in training. However, in practice, NEIL observes the training loss from only a finite set of m trajectories in each iteration. For this consideration, in the following discussion, we provide a proof of the cost bound of NEIL under the finite sample case.

Denote D_i as the m trajectories collected in the i-th iteration and $\ell_i(\hat{\pi}_i) = \mathbb{E}_{s\sim D_i}[\ell(s,\hat{\pi}_i)]$. Applying the no-regret assumption (Assumption A.1) allows us to bound the average expected policy training loss: $\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim D_i}\big[\ell(s,\pi_i)\big] - \tilde{\epsilon}_N \leq \tilde{\gamma}_N$,

where $\tilde{\epsilon}_N = \min_{\pi\in\Pi} \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{s\sim D_i}\big[\ell(s,\pi)\big]$ denotes the loss of the best policy in hindsight on the finite samples.

Following Eq. (5), we need to switch the derivation from the expected loss of $\hat{\pi}_i$ over d_{π_i} (i.e., $\mathbb{E}_{s\sim d_{\pi_i}}[\ell(s,\hat{\pi}_i)]$) to that over D_i (i.e., $\mathbb{E}_{s\sim D_i}[\ell(s,\hat{\pi}_i)]$), the actual state distribution that $\hat{\pi}_i$ is trained on. To fill this gap, we introduce Y_{ij} to denote the difference between the expected loss of $\hat{\pi}_i$ under d_{π_i} and the average loss of $\hat{\pi}_i$ under the j-th sample trajectory with π at iteration i. The random variables Y_{ij} over all $i \in \{1,2,...,N\}$ and $j \in \{1,2,...,m\}$ are all zero mean, bounded in $[-\ell_{max}, \ell_{max}]$ and form a martingale in the order of $Y_{11}, Y_{12}, ..., Y_{1m}, Y_{21}, ..., Y_{Nm}$. By Azuma-Hoeffding's inequality (Azuma, 1967; Hoeffding, 1994), $\frac{1}{mN}\sum_{i=1}^{N}\sum_{j=1}^{m} Y_{ij} \leq \ell_{max}\sqrt{\frac{2\log(1/\delta)}{mN}}$ with probability $1-\delta$. Following the derivations in Eq. (5) and by introducing Y_{ij}, with probability of $1-\delta$, we obtain the following inequalities by definition:

$$
\begin{aligned}
& J(\hat{\pi}) \\
\leq & \frac{T}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{s\sim d_{\pi_i}}[\ell(s,\hat{\pi}_i)] + \ell_{max}||d_{\pi_i} - d_{\pi^*}||_1\Big] \\
\leq & \frac{T}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{s\sim D_i}[\ell(s,\hat{\pi}_i)] + \ell_{max}||d_{\pi_i} - d_{\pi^*}||_1\Big] \\
& + \frac{T}{mN}\sum_{i=1}^{N}\sum_{j=1}^{m} Y_{ij} \\
\leq & \frac{T}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{s\sim D_i}[\ell(s,\hat{\pi}_i)] + \ell_{max}||d_{\pi_i} - d_{\pi^*}||_1\Big] \\
& + \ell_{max}T\sqrt{\frac{2\log(1/\delta)}{mN}} \\
\leq & T\Big[\tilde{\gamma}_N + \tilde{\epsilon}_N + \ell_{max}\sqrt{\frac{2\log(1/\delta)}{mN}} \\
& + \frac{2\ell_{max}T}{N}\sum_{i=1}^{N} e_i\Big].
\end{aligned}
$$

Notice that we need mN to be at least $\tilde{O}(T^2 log(1/\delta))$, so that $\tilde{\gamma_N}$ and $l_{max}\sqrt{\frac{2\log(1/\delta)}{mN}}$ are negligible. This leads to the following theorem:

Theorem A.3. *For the proposed* NEIL *algorithm, with probability at least* $1-\delta$*, when* mN *is* $\tilde{O}(T^2 log(1/\delta))$*, there exists a policy* $\hat{\pi} \in \hat{\pi}_{1:N}$ *s.t.* $J(\hat{\pi}) \leq T[\tilde{\epsilon}_N + \frac{2l_{max}T}{N}\sum_{i=1}^{N} e_i] + O(1)$.

The theorem shows that the cost of NEIL can still be bounded in the finite sample setting. Comparing this bound with the bound under the infinite sample setting, we can observe that the bound is still related to e_i, the probability that $\hat{\pi}_i$ takes a confident but incorrect action under d_{π^*}.

B Implementation Details

B.1 Interactive Semantic Parsing Framework

Our system assumes an interactive semantic parsing framework to collect user feedback. In experiments, this is implemented by adapting MISP (Yao et al., 2019b), an open-sourced framework[5] that has demonstrated a strong ability to improve test-time parsing accuracy. In this framework, an agent is comprised of three components: a world model that wraps the base semantic parser and a feedback incorporation module to interpret user feeds and update the semantic parse, an error detector that decides whether to request for user intervention, and an actuator that delivers the agent's request by asking a natural language question, such that users without domain expertise can understand.

We follow MISP's instantiation for text-to-SQL tasks to adopt a probability-based uncertainty estimator as the error detector, which triggers user interactions when the probability of the current decision is lower than a threshold. The actuator is instantiated by a grammar-based natural language generator. We use the latest version of MISP that allows multi-choice interactions to improve the system efficiency, i.e., when the parser's current decision is validated as wrong, the system presents multiple alternative options for user selection. An additional "None of the above options" option is included in case all top options from the system are wrong. Figure 1 shows an example of the user interaction. From there, the system can derive a correct decision to address its uncertainty (e.g., taking "`Player`" as a `WHERE` column).

As a general interactive semantic parsing framework, MISP has its advantage of being generalizable to different kinds of semantic parsers (as long as their parsing process can be formulated as taking a sequence of actions in their respective action space) and various logical forms (e.g., lambda expressions). Although it could be non-trivial to instantiate such an interactive system, we note that it is a one-time effort for all datasets of the same logical form.

Example of Non-sketch-based Parsers. In addi-

[5] https://github.com/sunlab-osu/MISP.

tion to the example of the SQLova parser (Hwang et al., 2019) that we provide in Section 3, here we show how the EditSQL parser (Zhang et al., 2019) is formulated under MISP. Unlike SQLova, EditSQL does not assume any SQL sketch; it instead generates a SQL query "token by token".[6] Consider the SQL query in Figure 1. EditSQL takes actions: a_1="`SELECT`", a_2="`COUNT`", a_3="`(`", a_4="`School/Club Team`", a_5="`)`", etc. Therefore, the action space of EditSQL consists of all SQL keywords, grammatical constituents (e.g., "`(`"), and available table columns. In this case, MISP only validates semantically meaningful actions (including aggregators, operators, column names, etc.) while skipping others (including trivial symbols like "`(`" and most SQL keywords[7]).

User Simulator. Our experiments train each system with simulated user feedback. To this end, we build a user simulator similar to the one used by Yao et al. (2019b) in MISP, which can access the ground-truth SQL queries. It gives yes/no answer or selects a choice by directly comparing the sampled policy action with the true one in the gold query. When the true option is not presented within the system provided choices, the user is simulated to select "None of the above options".

B.2 WikiSQL Experiment Details

Dataset&Model. Our main experiments consider the WikiSQL benchmark dataset (Zhong et al., 2017),[8] which contains 56,355/8,421/15,878 question-SQL query pairs in the training/validation/test set. We use exactly the same data split as Zhong et al. (2017).

We choose SQLova (Hwang et al., 2019), one of the open-sourced[9] top-performing semantic parser on WikiSQL, as the base parser, which ensures reasonable model capability to study continual learning. Hyper-parameters are set the same as the ones recommended by the SQLova authors on their GitHub repository,[10] except that we use a learning rate of 1e-5 for fine-tuning the BERT model. Empirically we found out this relatively larger learning rate can greatly accelerate the model learning without affecting the model performance significantly. The total number of model parameters is around 118M, with 110M from BERT-Base (Uncased)[11] and 8M from the SQLova parser side.

Early stop is used to accelerate model training in each training iteration. Specifically, we stop model training if it does not show improvement on the validation set for a consecutive number of epochs. We set this number to 10 before the 30-th training iteration when the total training data is in a relatively small size, and decay it to 5 after the 30-th iteration. We follow SQLova when preprocessing the WikiSQL data.

Experimental Setup. We study a "continual learning" problem and experiment various systems with three initialization settings, as suggested by our theoretical analysis (Section 5). Specifically, we use 10% (5,636 pairs), 5% (2,818 pairs), and 1% (564 pairs) of the total training data for parser initialization, respectively.

In each initialization setting, the remaining training data is used to simulate user questions that a system receives after deployment. The user questions come in a random order. We repeat three random runs (i.e., three random orders of user questions) and report the average system performance. Notice that, we ensure each system receive the same user question (but may have different user feedback depending on their interaction designs) during iterative training, for a fair comparison. Systems update (retrain) their base semantic parsers periodically for every 1,000 user questions.

Metrics. In the end of each iteration, we evaluate the system's performance, including:

- Parsing accuracy. We measure the query match accuracy (i.e., logical form accuracy) using the script from SQLova implementation.
- An accumulated number of user/expert annotations (introduced in Section 6.3). Different systems request different kinds of user/expert annotations. Therefore, even when serving the user on the same user question, different systems require different numbers of annotations. This metric sums up the total number of annotations that each system has requested after each training iteration.

Calculating the aforementioned metrics allow us to plot Figure 2 and Figure 4.

[6] EditSQL considers a column name as a single "token", although it may actually contain several words (e.g., `School/Club Team`).

[7] Except `WHERE`, `GROUP BY`, `ORDER BY` and `HAVING`; see Appendix B.3 for details.

[8] https://github.com/salesforce/WikiSQL.

[9] https://github.com/naver/sqlova.

[10] https://github.com/naver/sqlova#running-code.

[11] https://github.com/google-research/bert#pre-trained-models.

Compute. We complete experiments on Nvidia GeForce RTX 2080Ti (11GB). Models are all implemented using PyTorch.[12] The run time for each training iteration varies depending on the accumulated training data size. To finish the 50+ iterations of (re-)training, each system takes around 15 days. In the weak 1% initialization case, the Binary User baseline takes less time (around 10 days), since most of its predicted queries are wrong and thus are not included into its training data.

B.3 EditSQL Experiment Details

Data&Model. The Spider dataset (Yu et al., 2018) contains 8,421 question-SQL pairs for training and 1,034 pairs for validation.[13] The test set is not publicly available and is thus not used in our experiments.

We choose EditSQL (Zhang et al., 2019) as the base semantic parser,[14] since it is one of the open-sourced state-of-the-art models on Spider. All hyper-parameters are set following (Zhang et al., 2019). Pre-trained BERT model is also used. Totally there are around 120M parameters in the model, with 110M from the BERT-Base (Uncased)[15] and 10M from the EditSQL parser side. Early stop is additionally used to accelerate model training. Specifically, we stop model training when it does not show improvement on validation for 5 consecutive epochs.

In the data preprocessing step, EditSQL transforms each gold SQL query into a sequence of tokens, where the `From` clause is removed and each column `Col` is prepended by its paired table name, i.e., `Tab.Col`. However, we observe that sometimes this transformation is not convertible. For example, consider the question "what are the first name and last name of all candidates?" and its gold SQL query: "`SELECT T2.first_name , T2.last_name FROM candidates AS T1 JOIN people AS T2 ON T1.candidate_id = T2.person_id`". EditSQL transforms this query into : "`select people.first_name , people.last_name`". The transformed sequence accidentally removes the information about table `candidates` in the original SQL query, leading to semantic meaning inconsistent with the question. When using such erroneous sequences as the gold targets in model training, we cannot simulate consistent user feedback, e.g., when the user is asked whether her query is relevant to the table `candidates`, the simulated user cannot give an affirmative answer based on the transformed sequence. To avoid inconsistent user feedback, we remove question-SQL pairs whose transformed sequence is inconsistent with the original gold SQL query, from the training data. This can be easily done by using EditSQL's post-processing script to convert a preprocessed sequence back to the SQL format. Only when the converted query is the same as the original one, the transformation is consistent. This reduces the size of the training set from 8,421 to 7,377. The validation set is kept untouched for fair evaluation.

The implementation of interactive semantic parsing for EditSQL is the same as Section B.1, except that, in order to cope with the complicated structure of Spider SQL queries, for columns in `WHERE`, `GROUP BY`, `ORDER BY` and `HAVING` clauses, we additionally provide an option for the user to "remove" the clause, e.g., removing a `WHERE` clause by picking the "The system does not need to consider any conditions." option. We also adjust the "semantic unit" definition in MISP[16] to deal with the autoregressive decoding of EditSQL. For example, instead of asking first about a `SELECT` column and then about its aggregator, we define one semantic unit to inquire about both the column and its aggregator.

To instantiate NEIL, the confidence threshold μ is 0.995 as we observe that EditSQL tends to be overconfident.

Experimental Setup. We experiment with one initialization setting, using 10% of the total training data (i.e., 737 question-SQL pairs), and systems update (retrain) their base semantic parsers periodically for every 1,000 user questions as in WikiSQL experiments. We report system performance averaged over three random runs (i.e., three random orders of user questions).

We also tried using more training data for initialization. However, since the total training data in Spider is very limited in size, more initialization data means fewer data for simulating online

[12] https://pytorch.org/.

[13] https://github.com/taoyds/spider.

[14] https://github.com/ryanzhumich/editsql.

[15] https://github.com/google-research/bert#pre-trained-models.

[16] https://github.com/sunlab-osu/MISP/blob/multichoice_q/MISP_SQL/tag_seq_logic.md.

user questions and conducting continual learning. This leads to less clear experimental observations (e.g., even the Full Expert system shows fluctuation, probably due to data redundancy or an issue with model architecture capability). Therefore, we only focus on the 10% initialization setting.

Metrics. We measure each system similarly as in WikiSQL experiments. For parsing performance, we calculate the exact match accuracy using scripts from the EditSQL implementation.

Compute. We complete experiments on Nvidia GeForce RTX 2080Ti (11GB). Models are implemented using PyTorch. The run time for each training iteration varies depending on the accumulated training data size. Finishing the whole iterative learning takes around 5 days for all systems.

C Additional Experimental Results

C.1 Additional SQLova Results

Figure 4 shows different systems' performance on WikiSQL validation set. For Binary User(+Expert), it is hard to quantify "one annotation", which varies according to the actual database size and the query difficulty. As a compromise, we approximate this number by calculating it in the same way as Full Expert, with the assumption that in general validating execution results is as hard as validating the SQL query itself.

We also show in Figure 5 the average number of annotations (i.e., user interactions) that MISP-NEIL requires per question during the iterative training. Overall, as the base parser is further trained, our system tends to request fewer user interactions. In most cases throughout the training, the system requests no more than one user interaction, demonstrating the annotation efficiency of our NEIL algorithm.

C.2 Connection to Theoretical Analysis

As we proved in Section 5, the performance gap between our proposed NEIL algorithm and the supervised approach is mainly decided by $\frac{1}{N}\sum_{i=1}^{N} e_i$, an average probability that $\hat{\pi}_i$ makes a confident but wrong decision under d_{π^*} (i.e., given a gold partial parse) over N training iterations. More specifically, from our proof of Lemma A.1, e_i can be expressed as:

$$e_i = \frac{1}{T}\sum_{t=1}^{T} e_{it} = \frac{1}{T}\sum_{t=1}^{T} \tilde{\epsilon}_{it}(1-\beta_{it}),$$

where $\tilde{\epsilon}_{it}$ denotes policy $\hat{\pi}_i$'s conditional error rate under $d^t_{\pi^*}$ when it does not query the expert (i.e., being confident about its own action) at step t, and $1-\beta_{it}$ denotes the probability that $\hat{\pi}_i$ does not query the expert under $d^t_{\pi^*}$. $\tilde{\epsilon}_{it}(1-\beta_{it})$ thus represents a joint probability that $\hat{\pi}_i$ makes confident but wrong action under $d^t_{\pi^*}$ at step t.

To show a reflection of our theoretical analysis on the experiments, we present the values of the following three variables during training: (1) $\tilde{\epsilon}_i = \frac{1}{T}\sum_{t=1}^{T}\tilde{\epsilon}_{it}$, the average value of $\tilde{\epsilon}_{it}$ over T time steps. A smaller $\tilde{\epsilon}_i$ implies a lower conditional error rate and thus a smaller e_i and a smaller performance gap. (2) $\beta_i = \frac{1}{T}\sum_{t=1}^{T}\beta_{it}$, the average value of β_{it} over T time steps. A smaller β_i (i.e., a larger $1-\beta_i$) means a smaller probability that $\hat{\pi}_i$ queries the expert (i.e., being more confident). This could lead to a larger e_i and thus a larger performance gap. (3) e_i as defined previously. A smaller e_i indicates a smaller performance gap between our algorithm and the supervised approach.

We plot the results of our MISP-NEIL system (based on SQLova) in Figure 6. For all initialization settings, we observe that the base parser tends to make more confident actions under a gold partial parse (i.e., decreasing β_i) when it is trained for more iterations. Meanwhile, the error rate of its confident actions under a gold partial parse is also reduced (i.e., decreasing $\tilde{\epsilon}_i$). When combining the two factors, e_i is shown to keep decreasing, implying that with more iterations that the parser is trained, it gets a tighter cost bound and better performance.

Finally, we notice that a differently initialized parser can end up with different performance. This is reasonable since a better initialized parser presumably should have a better overall error rate. This is also consistent with our observation in the main experimental results (Section 6.3).

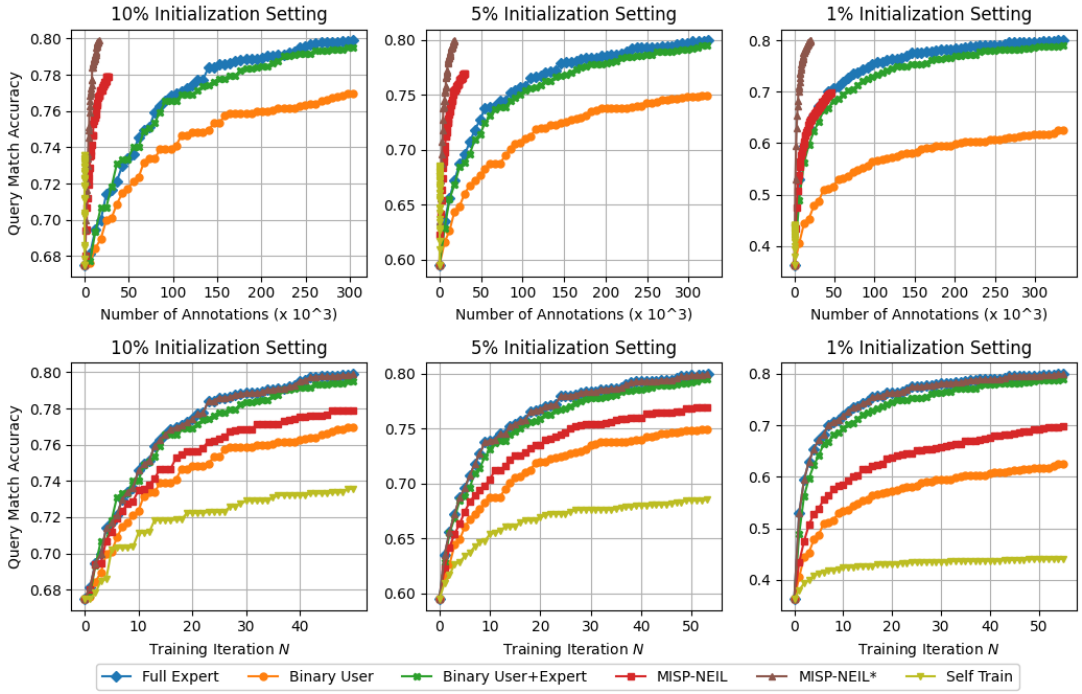


Figure 4: Parsing accuracy on WikiSQL validation set when systems are trained with various numbers of user/expert annotations (top) and for different iterations (bottom). We experiment systems with three initialization settings, using 10%, 5% and 1% of the training data respectively.

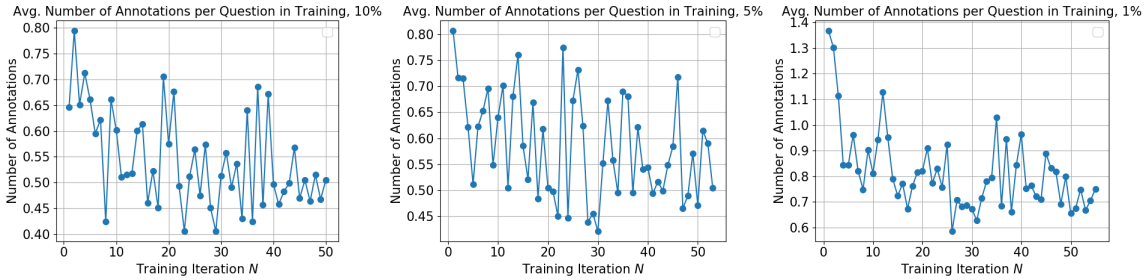
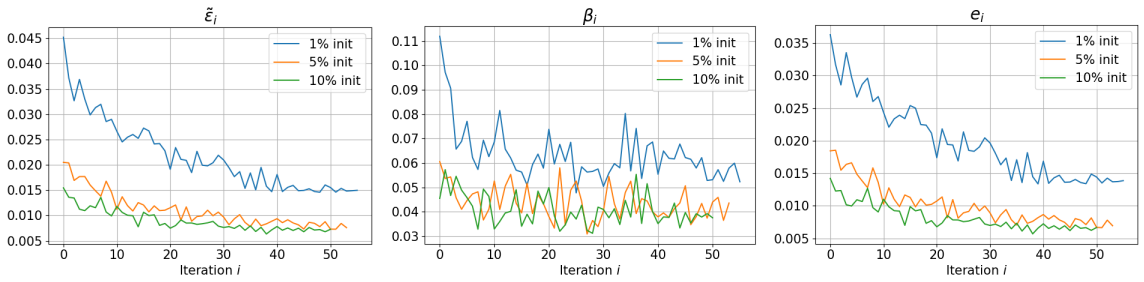


Figure 5: Average number of user annotations/interactions that MISP-NEIL requests for each user question during iterative training (on WikiSQL), when the parser is initialized using 10%, 5% and 1% of training data.



(a) (b) (c)

Figure 6: The values of $\tilde{\epsilon}_i$ (a), β_i (b) and e_i (c) in MISP-NEIL throughout the training (on WikiSQL validation set), under different initialization settings.

Rationalizing Medical Relation Prediction from Corpus-level Statistics

Zhen Wang[1], **Jennifer Lee**[2,3], **Simon Lin**[4], **Huan Sun**[1]
[1]The Ohio State University
[2]Department of Family Medicine, The Ohio State University Wexner Medical Center
[3]Department of Physician Informatics, Nationwide Childrens Hospital
[4]Abigail Wexner Research Institute at Nationwide Children's Hospital
{wang.9215, sun.397}@osu.edu
{Jennifer.Lee2, Simon.Lin}@nationwidechildrens.org

Abstract

Nowadays, the interpretability of machine learning models is becoming increasingly important, especially in the medical domain. Aiming to shed some light on how to rationalize medical relation prediction, we present a new interpretable framework inspired by existing theories on how human memory works, e.g., theories of recall and recognition. Given the corpus-level statistics, i.e., a global co-occurrence graph of a clinical text corpus, to predict the relations between two entities, we first *recall* rich contexts associated with the target entities, and then *recognize* relational interactions between these contexts to form model rationales, which will contribute to the final prediction. We conduct experiments on a real-world public clinical dataset and show that our framework can not only achieve competitive predictive performance against a comprehensive list of neural baseline models, but also present rationales to justify its prediction. We further collaborate with medical experts deeply to verify the usefulness of our model rationales for clinical decision making. Code and datasets are available online[1].

1 Introduction

Predicting relations between entities from a text corpus is a crucial task in order to extract structured knowledge, which can empower a broad range of downstream tasks, e.g., question answering (Xu et al., 2016), dialogue systems (Lowe et al., 2015), reasoning (Das et al., 2017), etc. There has been a large amount of existing work focusing on predicting relations based on *raw texts* (e.g., sentences, paragraphs) mentioning two entities (Hendrickx et al., 2010; Zeng et al., 2014; Zhou et al., 2016; Mintz et al., 2009; Riedel et al., 2010; Lin et al., 2016; Verga et al., 2018; Yao et al., 2019).

[1]https://github.com/zhenwang9102/X-MedRELA

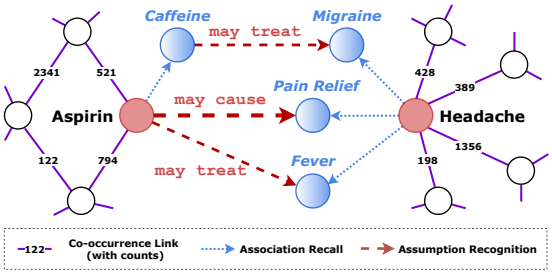


Figure 1: Our intuition for how to rationalize relation prediction based on the corpus-level statistics. To infer the relation between the target entities (red nodes), we recall (blue dashed line) their associated entities (blue nodes) and infer their relational interactions (red dashed line), which will serve as assumptions or model rationales to support the target relation prediction.

In this paper, we study a relatively new setting in which we predict relations between entities based on the *global co-occurrence statistics* aggregated from a text corpus, and focus on medical relations and clinical texts in Electronic Medical Records (EMRs). The corpus-level statistics present a *holistic graph view* of all entities in the corpus, which will greatly facilitate the relation inference, and can better preserve patient privacy than raw or even de-identified textual content and are becoming a popular substitute for the latter in the research community for studying EMR data (Finlayson et al., 2014; Wang et al., 2019).

To predict relations between entities based on a global co-occurrence graph, intuitively, one can first optimize the graph embedding or global word embedding (Pennington et al., 2014; Perozzi et al., 2014; Tang et al., 2015), and then develop a relation classifier (Nickel et al., 2011; Socher et al., 2013; Yang et al., 2015; Wang et al., 2018) based on the embedding vectors of the two entities. However, such kind of neural frameworks often lack the desired *interpretability*, which is especially important for the medical domain. In general, despite their superior predictive performance in many NLP

tasks, the opaque decision-making process of neural models has concerned their adoption in high stakes domains like medicine, finance, and judiciary (Rudin, 2019; Murdoch et al., 2019). Building models that provide reasonable explanations and have increased transparency can remarkably enhance user trust (Ribeiro et al., 2016; Miller, 2019). In this paper, we aim to develop such a model for our medical relation prediction task.

To start with, we draw inspiration from the existing theories on cognitive processes about how human memory works, e.g., two types of memory retrieval (recall and recognition) (Gillund and Shiffrin, 1984). Basically, in the *recall* process, humans tend to retrieve contextual associations from long-term memory. For example, given the word "Paris", one may think of "Eiffel Tower" or "France", which are strongly associated with "Paris" (Nobel and Shiffrin, 2001; Kahana et al., 2008; Budiu, 2014). Besides, there is a strong correlation between the association strength and the co-occurrence graph (Spence and Owens, 1990; Lundberg and Lee, 2017). In the *recognition* process, humans typically recognize if they have seen a certain piece of information before. Figure 1 shows an example in the context of relation prediction. Assume a model is to predict whether *Aspirin* may treat *Headache* or not (That "*Aspirin* may treat *Headache*" is a known fact, and we choose this relation triple for illustration purposes). It is desirable if the model could perform the aforementioned two types of memory processes and produce rationales to base its prediction upon: (1) Recall. What entities are associated with *Aspirin*? What entities are associated with *Headache*? (2) Recognition. Do those associated entities hold certain relations, which can be leveraged as clues to predict the target relation? For instance, a model could first retrieve a relevant entity *Pain Relief* for the tail entity *Headache* as they co-occur frequently, and then recognize there is a chance that *Aspirin* can lead to *Pain Relief* (i.e., formulate model rationales or assumptions), based on which it could finally make a correct prediction (*Aspirin*, may treat, *Headache*).

Now we formalize such intuition to rationalize the relation prediction task. Our framework consists of three stages, *global association recall* (CogStage-1), *assumption formation and representation* (CogStage-2), and *prediction decision making* (CogStage-3), shown in Figure 2. CogStage-1 models the process of recalling diverse contextual

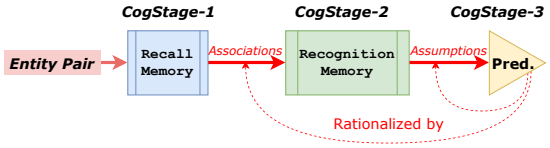


Figure 2: A high-level illustration of our framework.

entities associated with the target head and tail entities respectively, CogStage-2 models the process of recognizing possible interactions between those recalled entities, which serve as model rationales (or, assumptions[2]) and are represented as semantic vectors, and finally CogStage-3 aggregates all assumptions to infer the target relation. We jointly optimize all three stages using a training set of relation triples as well as the co-occurrence graph. Model rationales can be captured through this process *without any gold rationales* available as direct supervision. Overall, our framework rationalizes its relation prediction and is interpretable to users[3] by providing justifications for (i) why a particular prediction is made, (ii) how the assumptions of the prediction are developed, and (iii) how the particular assumptions are relied on.

On a real-life clinical text corpus, we compare our framework with various competitive methods to evaluate the predictive performance and interpretability. We show that our method obtains very competitive performance compared with a comprehensive list of various neural baseline models. Moreover, we follow recent work (Singh et al., 2019; Jin et al., 2020) to quantitatively evaluate model interpretability and demonstrate that rationales produced by our framework can greatly help earn expert trust. To summarize, we study the important problem of rationalizing medical relation prediction based on corpus-level statistics and propose a new framework inspired by cognitive theories, which outperforms competitive baselines in terms of both interpretability and predictive performance.

2 Background

Different from existing work using raw texts for relation extraction, we assume a global co-occurrence graph (i.e., corpus-level statistics) is given, which was pre-constructed based on a text corpus $\mathcal{D}$, and denote it as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each vertex $v \in \mathcal{V}$ represents an entity extracted

[2]We use the two terms interchangeably in this paper.

[3]Following Murdoch et al. (2019), desired interpretability is supposed to provide insights to particular audiences, which in our case are medical experts.

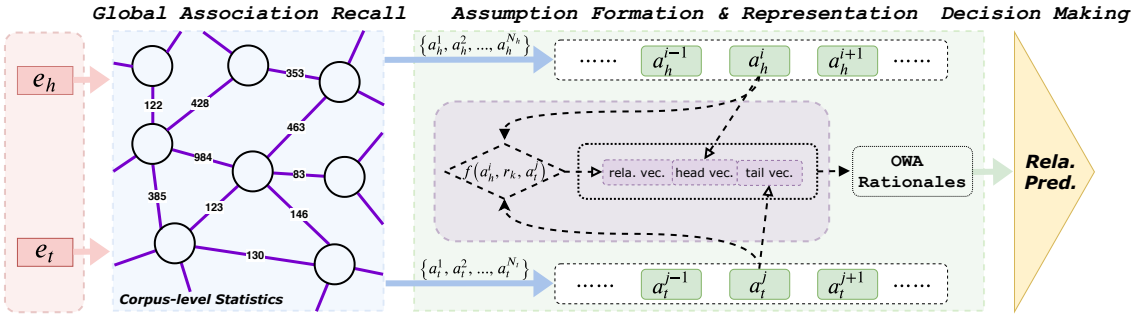


Figure 3: Framework Overview.

from the corpus and each edge $e \in \mathcal{E}$ is associated with the global co-occurrence count for the connected nodes. Counts reflect how frequent two entities appear in the same context (e.g., co-occur in the same sentence, document, or a certain time frame). In this paper, we focus on clinical co-occurrence graph in which vertices are medical terms extracted from clinical notes. Nevertheless, as we will see later, our framework is very general and can be applied to other relations with corpus-level statistics.

Our motivation for working under this setting lies in three folds: (1) Such graph data is stripped of raw textual contexts and thus, has a better preserving of patient privacy (Wang et al., 2019), which makes itself easier to be constructed and shared under the HIPPA protected environments (Act, 1996) for medical institutes (Finlayson et al., 2014); (2) Compared with open-domain relation extraction, entities holding a medical relation oftentimes do not co-occur in a local context (e.g., a sentence or paragraph). For instance, we observe that in a widely used clinical co-occurrence graph (Finlayson et al., 2014), which is also employed for our experiments later, of all entity pairs holding the treatment relation according to UMLS (Unified Medical Language System), only about 11.4% have a co-occurrence link (i.e., co-occur in clinical notes within a time frame like 1 day or 7 days); (3) As suggested by cognitive theories (Spence and Owens, 1990), lexical co-occurrence is significantly correlated with association strength in the recall memory process, which further inspires us to utilize such statistics to find associations and form model rationales for relation prediction.

Finally, our relation prediction task is formulated as: Given the global statistics $\mathcal{G}$ and an entity pair, we predict whether they hold a relation r (e.g., MAY_TREAT), and moreover provide a set of model rationales $\mathcal{T}$ composed of relation triples for the prediction. For the example in Figure 1, we aim to build a model that will not only accurately predict the MAY_TREAT relation, but also provide meaningful rationales on how the prediction is made, which are crucial for gaining trust from clinicians.

3 Methodology

Following a high-level framework illustration in Figure 2, we show a more detailed overview in Figure 3 and introduce each component as follows.

3.1 CogStage-1: Global Association Recall

Existing cognitive theories (Kahana et al., 2008) suggest that *recall* is an essential function of human memory to retrieve *associations* for later decision making. On the other hand, the association has been shown to significantly correlate with the lexical co-occurrence from the text corpus (Spence and Owens, 1990; Lund and Burgess, 1996). Inspired by such theories and correlation, we explicitly build up our model based on recalled associations stemming from corpus-level statistics and provide global highly-associated contexts as the source of interpretations.

Given an entity, we build an estimation module to globally infer associations based on the corpus-level statistics. Our module leverages distributional learning to fully explore the graph structure. One can also directly utilize the raw neighborhoods in the co-occurrence graph, but due to the noise introduced in the preprocessing of building the graph, it is a less optimal choice in real practice.

Specifically, for a selected node/entity $e_i \in \mathcal{E}$, our global association recall module estimates a conditional probability $p(e_j|e_i)$, representing how likely the entity $e_j \in \mathcal{E}$ is associated with e_i[4]. We formally define such conditional probability as:

$$p(e_j|e_i) = \frac{\exp(\boldsymbol{v}'^T_{e_j} \cdot \boldsymbol{v}_{e_i})}{\sum_{k=1}^{|\mathcal{V}|} \exp(\boldsymbol{v}'^T_{e_k} \cdot \boldsymbol{v}_{e_i})} \quad (1)$$

[4]We assume all existing entities can be possible associations for the given entity.

where $\boldsymbol{v}_{e_i} \in \mathbb{R}^d$ is the embedding vector of node e_i and $\boldsymbol{v}'_{e_j} \in \mathbb{R}^d$ is the context embedding for e_j.

There are many ways to approximate $p\left(e_j|e_i\right)$ from the global statistics, e.g., using global log-bilinear regression (Pennington et al., 2014). To estimate such probabilities and update entity embeddings efficiently, we optimize the conditional distribution $p\left(e_j|e_i\right)$ to be close to the empirical distribution $\hat{p}\left(e_j|e_i\right)$ defined as:

$$\hat{p}\left(e_j|e_i\right) = \frac{p_{ij}}{\sum_{(i,k)\in\mathcal{E}} p_{ik}} \tag{2}$$

where $\mathcal{E}$ is the set of edges in the co-occurrence graph and p_{ij} is the PPMI value calculated by the co-occurrence counts between node e_i and e_j. We adopt the cross entropy loss for the optimization:

$$\mathcal{L}_n = -\sum_{(e_i,e_j)\in\mathcal{V}} \hat{p}(e_j|e_i)\log\left(p(e_j|e_i)\right) \tag{3}$$

This association recall module will be jointly trained with other objective functions to be introduced in the following sections. After that, given an entity e_i, we can select the top-N_c entities from $p(\cdot|e_i)$ as e_i's associative entities for subsequent assumption formation.

3.2 CogStage-2: Assumption Formation and Representation

As shown in Figure 3, with the associative entities from CogStage-1, we are ready to *formulate* and *represent* assumptions. In this paper, we define model assumptions as *relational interactions between associations*, that is, as shown in Figure 1, the model may identify (*Caffeine*, MAY_TREAT, *Migraine*) as an assumption, which could help predict *Aspirin* may treat *Headache* (*Caffeine* and *Migraine* are associations for *Aspirin* and *Headache* respectively). Such relational rationales are more concrete and much easier for humans to understand than the widely-adopted explanation strategy (Yang et al., 2016; Mullenbach et al., 2018; Vashishth et al., 2019) in NLP that is based on pure attention weights on local contexts.

One straightway way to obtain such rationales is to query existing medical knowledge bases (KBs), e.g., (*Caffeine*, MAY_TREAT, *Migraine*) may exist in SNOMED CT[5] and can serve as a model rationale. We refer to rationales acquired in this way as the *Closed-World Assumption* (CWA) (Reiter, 1981) setting since only KB-stored facts are considered and trusted in a closed world. In contrast to the CWA rationales, considering the sparsity and incompleteness issues of KBs that are even more severe in the medical domain, we also propose the *Open-World Assumptions* (OWA) (Ceylan et al., 2016) setting to discover richer rationales by estimating all potential relations between associative entities based on a seed set of relation triples (which can be regarded as prior knowledge).

In general, the CWA rationales are relatively more accurate as each fact triple has been verified by the KB, but would have a low coverage of other possibly relevant rationales for the target prediction. On the other hand, the OWA rationales are more comprehensive but could be noisy and less accurate, due to the probabilistic estimation procedure and the limited amount of prior knowledge. However, as we will see, by aggregating all OWA rationales into the whole framework with an attention-based mechanism, we can select high-quality and most relevant rationales for prediction. For the rest of the paper, by default we adopt the OWA setting in our framework and describe its details as follows.

Specifically, given a pair of head and tail entity, $e_h, e_t \in \mathcal{V}$, let us denote their association sets as $\mathcal{A}(e_h) = \{a_h^i\}_{i=1}^{N_h}$ and $\mathcal{A}(e_t) = \{a_t^j\}_{j=1}^{N_t}$, where N_h, N_t are the number of associative entities a_h, a_t to use. Each entity has been assigned an embedding vector by the previous association recall module. We first measure the probability of relations holding for the pair. Given $a_h^i \in \mathcal{A}(e_h), a_t^j \in \mathcal{A}(e_t)$ and a relation $r_k \in \mathcal{R}$, we define a scoring function as Bordes et al. (2013) to estimate triple quality:

$$s_k^{ij} = f(a_h^i, r_k, a_t^j) = -||\boldsymbol{v}_{a_h^i} + \boldsymbol{\xi}_k - \boldsymbol{v}_{a_t^j}||_1 \tag{4}$$

where $\boldsymbol{v}_{a_h^i}$ and $\boldsymbol{v}_{a_t^j}$ are embedding vectors, relations are parameterized by a relation matrix $R \in \mathbb{R}^{N_r \times d}$ and $\boldsymbol{\xi}_k$ is its k-th row vector. Such a scoring function encourages larger value for correct triples. Additionally, in order to filter unreliable estimations, we define an NA relation to represent other trivial relations or no relation as the score, $s_{\text{NA}}^{ij} = f(a_h^i, \text{NA}, a_t^j)$, which can be seen as a dynamic threshold to produce reasonable rationales.

Now we *formulate* OWA rationales by calculating the conditional probability of a relation given a pair of associations as follows (we save the superscript ij for space):

$$p(r_k|a_h^i, a_t^j) = \begin{cases} \dfrac{\exp\left(s_k\right)}{\sum_{s_k \geq s_{\text{NA}}} \exp\left(s_k\right)}, & s_k > s_{\text{NA}} \\ 0, & s_k \leq s_{\text{NA}} \end{cases} \tag{5}$$

[5] https://www.snomed.org/

For each association pair, (a_h^i, a_t^j), we only form an assumption with a relation r_k^* if r_k^* is top ranked according to $p(r_k|a_h^i, a_t^j)$[6].

To *represent* assumptions, we integrate all relation information per pair into a single vector representation. Concretely, we calculate the assumption representation by treating $p(r_k|a_h^i, a_t^j)$ as weights for all relations as follows:

$$\mathsf{a}_{ij} = \rho(a_h^i, a_t^j; \mathcal{R}) = \sum_{k'=1}^{N_r} p(r_{k'}|a_h^i, a_t^j) \cdot \boldsymbol{\xi}_{k'} \quad (6)$$

Finally, we combine the entity vectors as well as the relation vector to get the final representation of assumptions for association pair (a_h^i, a_t^j), where $c_i \in \mathcal{A}(e_h)$ and $c_j \in \mathcal{A}(e_t)$:

$$\mathsf{e}_{ij} = \tanh([\boldsymbol{v}_{a_h^i}; \boldsymbol{v}_{a_t^j}; \mathsf{a}_{ij}]\boldsymbol{W}_p + \boldsymbol{b}_p) \quad (7)$$

where $[\cdot\,;\cdot]$ represents vector concatenation, $\boldsymbol{W}_p \in \mathbb{R}^{3d\times d_p}$, $\boldsymbol{b}_p \in \mathbb{R}^{d_p}$ are the weight matrix and bias in a fully-connected network.

3.3 CogStage-3: Prediction Decision Making

Analogical to human thinking, our decision making module aggregates all assumption representations and measures their accountability for the final prediction. It learns a distribution over all assumptions and we select the ones with highest probabilities as model rationales. More specifically, we define a scoring function $g(\mathsf{e}_{ij})$ to estimate the accountability based on the assumption representation e_{ij} and normalize $g(\mathsf{e}_{ij})$ as:

$$g(\mathsf{e}_{ij}) = \boldsymbol{v}^T \cdot \tanh(\boldsymbol{W}_a \mathsf{e}_{ij} + \boldsymbol{b}_a) \quad (8)$$

$$p_{ij} = \frac{\exp(g(\mathsf{e}_{ij}))}{\sum_{m=1}^{N_h}\sum_{n=1}^{N_t}\exp(g(\mathsf{e}_{mn}))} \quad (9)$$

where $\boldsymbol{W}_a, \boldsymbol{b}_a$ are the weight matrix and bias for the scoring function. Then we get the weighted rationale representation as:

$$\mathsf{r} = \psi(e_h, e_t) = \sum_{i=1}^{N_h}\sum_{j=1}^{N_t} p_{ij}\mathsf{e}_{ij} \quad (10)$$

With the representation of weighted assumption information for the target pair (e_h, e_t), we calculate the binary prediction probability for relation r as:

$$p(r|e_h, e_t) = \sigma(\boldsymbol{W}_r \mathsf{r} + \boldsymbol{b}_r) \quad (11)$$

where $\sigma(x) = 1/(1+\exp(-x))$ and $\boldsymbol{W}_r, \boldsymbol{b}_r$ are model parameters.

[6]We remove the target relation to predict if it exists in the assumption set.

Rationalizing relation prediction. After fully training the entire model, to recover the most contributing assumptions for predicting the relation between the given target entities (e_h, e_t), we compute the importance scores for all assumptions and select those most important ones as model rationales. In particular, we multiply p_{ij} (the weight for association pair (a_h^i, a_t^j) in Eqn. 9) with $p(r_k|a_h^i, a_t^j)$ (the probability of a relation given the pair (a_h^i, a_t^j) in Eqn. 5) to score the triple (a_h^i, r_k, a_t^j). We rank all such triples for $a_h^i \in \mathcal{A}(e_h), a_t^j \in \mathcal{A}(e_t), r_k \in \mathcal{R}$ and select the top-K triples as model rationales for the final relation prediction.

3.4 Training

We now describe how we train our model efficiently for multiple modules. For relational learning to estimate the conditional probability $p(r_k|a_h^i, a_t^j)$, we utilize training data as the seed set of triples for all relations as correct triples denoted as $(h, r, t) \in \mathcal{P}$. The scoring function in Eqn. 4 is expected to score higher for correct triples than the corrupted ones in which we denote $\mathcal{N}(?, r, t)$ ($\mathcal{N}(t, r, ?)$) as the set of corrupted triples by replacing the head (tail) entity randomly. Instead of using margin-based loss function, we adopt a more efficient training strategy from (Kadlec et al., 2017; Toutanova and Chen, 2015) with a negative log likelihood loss function as:

$$\begin{aligned}\mathcal{L}_r = &-\textstyle\sum_{(h,r,t)\in\mathcal{P}} \log p\,(h|t,r) \\ &-\textstyle\sum_{(h,r,t)\in\mathcal{P}} \log p\,(t|h,r)\end{aligned} \quad (12)$$

where the conditional probability $p(h|t,r)$ is defined as follows ($p(t|h,r)$ is defined similarly):

$$p\,(h|t,r) = \frac{\exp(f\,(h,r,t))}{\sum_{h'\in\mathcal{N}(?,r,t)}\exp(f\,(h',r,t))} \quad (13)$$

For our binary relation prediction task, we define a binary cross entropy loss function with Eqn. 11 as follows:

$$\begin{aligned}\mathcal{L}_p = -\textstyle\sum_{i=1}^{M}&(y_i \cdot \log(p(r|e_h^i, e_t^i)) \\ &+ (1-y_i)\cdot\log(1-p(r|e_h^i, e_t^i)))\end{aligned} \quad (14)$$

where M is the number of samples, y_i is the label showing whether e_h, e_t holds a certain relation.

The above three loss functions, i.e., $\mathcal{L}_n$ for global association recall, $\mathcal{L}_r$ for relational learning and $\mathcal{L}_p$ for relation prediction, are all jointly optimized. All three of them share the entity embeddings and $\mathcal{L}_p$ will reuse the relation matrix from $\mathcal{L}_r$ to conduct the rationale generation. Please see appendix for more details of our training algorithm.

4 Experiments

In this section, we first introduce our experimental setup, e.g, the corpus-level co-occurrence statistics and datasets used for our experiments, and then compare our model with a list of comprehensive competitive baselines in terms of predictive performance. Moreover, we conduct expert evaluations as well as case studies to demonstrate the usefulness of our model rationales.

4.1 Dataset

We directly adopt a publicly available medical co-occurrence graph for our experiments (Finlayson et al., 2014). The graph was constructed in the following way: Finlayson et al. (2014) first used an efficient annotation tool (LePendu et al., 2012) to extract medical terms from 20 million clinical notes collected by Stanford Hospitals and Clinics, and then computed the co-occurrence counts of two terms based on their appearances in one patient's records within a certain time frame (e.g., 1 day, 7 days). We experiment with their biggest dataset with the largest number of nodes (i.e., the per-bin 1-day graph here[7]) so as to have sufficient training data. The co-occurrence graph contains 52,804 nodes and 16,197,319 edges.

To obtain training labels for relation prediction, we utilize the mapping between medical terms and concepts provided by Finlayson et al. (2014). To be specific, they mapped extracted terms to UMLS concepts with a high mapping accuracy by suppressing the least possible meanings of each term (see Finlayson et al. (2014) for more details). We utilize such mappings to automatically collect relation labels from UMLS. For term e_a and e_b that are respectively mapped to medical concept c_A and c_B, we find the relation between c_A and c_B in UMLS, which will be used as the label for e_a and e_b.

Following Wang and Fan (2014) that studied distant supervision in medical text and identified several crucial relations for clinical decision making, we select 5 important medical relations with no less than 1,000 relation triples in our dataset. Each relation is mapped to UMLS semantic relations, e.g., relation CAUSES corresponds to *cause_of*, *induces*, *causative_agent_of* in UMLS. A full list of mapping is in the appendix. We sample an equal number of negative pairs by randomly pairing head and tail entities with the correct argument types (Wang et al., 2016). We split all samples into train/dev/test sets with a ratio of 70/15/15. Only relation triples in the training set are used to optimize relational parameters. The statistics of the positive samples for relations are summarized in Table 1.

Med Relations	Train	Dev	Test
Symptom of	14,326	3,001	3,087
May treat	12,924	2,664	2,735
Contraindicates	10,593	2,237	2,197
May prevent	2,113	440	460
Causes	1,389	305	354
Total	41.3k	8.6k	8.8k

Table 1: Dataset Statistics.

4.2 Predictive Performance Evaluation

Compared Methods. There are a number of advanced neural methods (Tang et al., 2015; Qu et al., 2018; Wang et al., 2018) that have been developed for the link prediction task, i.e., predicting the relation between two nodes in a co-occurrence graph. At the high level, their frameworks comprise of an entity encoder and a relation scoring function. We adapt various existing methods for both the encoder and the scoring functions for comprehensive comparison. Specifically, given the co-occurrence graph, we employ existing distributional representation learning methods to learn entity embeddings. With the entity embeddings as input features, we adapt various models from the knowledge base completion literature as a binary relation classifier. More specifically, for the encoder, we select one word embedding method, Word2vec (Mikolov et al., 2013; Levy and Goldberg, 2014), two graph embedding methods, random-walk based DeepWalk (Perozzi et al., 2014), edge-sampling based LINE (Tang et al., 2015), and one distributional approach REPEL-D (Qu et al., 2018) for weakly-supervised relation extraction that leverages both the co-occurrence graph and training relation triples to learn entity representations. For the scoring functions, we choose DistMult (Yang et al., 2015), RESCAL (Nickel et al., 2011) and NTN (Socher et al., 2013).

Note that one can apply more complex encoders or scoring functions to obtain higher predictive performance; however, in this work, we emphasize more on model interpretability than predictive performance, and unfortunately, all such frameworks are hard to interpret as they provide little or no explanations on how predictions are made.

[7] https://datadryad.org/stash/dataset/doi:10.5061/dryad.jp917

Methods	MAY_TREAT	CONTRAIN.	SYMPTOM_OF	MAY_PREVENT	CAUSES	Avg.
Word2vec + DistMult	0.767 (±0.008)	0.777 (±0.013)	0.815 (±0.005)	0.649 (±0.018)	0.671 (±0.015)	0.736
Word2vec + RESCAL	0.743 (±0.010)	0.767 (±0.003)	0.808 (±0.009)	0.658 (±0.023)	0.659 (±0.039)	0.727
Word2vec + NTN	0.693 (±0.013)	0.758 (±0.005)	0.808 (±0.004)	0.605 (±0.022)	0.631 (±0.017)	0.699
DeepWalk + DistMult	0.740 (±0.003)	0.776 (±0.004)	0.805 (±0.003)	0.608 (±0.014)	0.650 (±0.018)	0.716
DeepWalk + RESCAL	0.671 (±0.010)	0.778 (±0.003)	0.800 (±0.003)	0.600 (±0.023)	**0.708 (±0.011)**	0.711
DeepWalk + NTN	0.696 (±0.006)	0.778 (±0.005)	0.787 (±0.005)	0.614 (±0.016)	0.674 (±0.024)	0.710
LINE + DistMult	0.767 (±0.003)	0.783 (±0.002)	0.795 (±0.003)	0.621 (±0.015)	0.641 (±0.024)	0.721
LINE + RESCAL	0.725 (±0.003)	0.771 (±0.002)	0.801 (±0.001)	0.613 (±0.013)	0.694 (±0.015)	0.721
LINE + NTN	0.733 (±0.002)	0.773 (±0.003)	0.800 (±0.001)	0.601 (±0.015)	0.706 (±0.013)	0.723
REPEL-D + DistMult	0.784 (±0.002)	0.797 (±0.002)	0.809 (±0.003)	0.681 (±0.010)	0.694 (±0.022)	0.751
REPEL-D + RESCAL	0.726 (±0.003)	0.780 (±0.002)	0.776 (±0.002)	**0.685 (±0.010)**	**0.708 (±0.003)**	0.737
REPEL-D + NTN	0.736 (±0.004)	0.780 (±0.002)	0.773 (±0.001)	0.667 (±0.015)	0.694 (±0.024)	0.731
Ours (w/ CWA)	0.709 (±0.005)	0.751 (±0.009)	0.744 (±0.007)	0.667 (±0.008)	0.661 (±0.032)	0.706
Ours	**0.805 (±0.017)**	**0.811 (±0.006)**	**0.816 (±0.004)**	0.676 (±0.020)	0.684 (±0.017)	**0.758**

Table 2: Comparison of model predictive performance. We run all methods for five times and report the averaged F1 scores with standard deviations.

We also show the predictive performance of our framework under the CWA setting in which the CWA rationales are existing triples in a "closed" knowledge base (i.e., UMLS). We first adopt the pre-trained association recall module to retrieve associative contexts for head and tail entities, then formulate the assumptions using top-ranked triples (that exist in our relation training data), where the rank is based on the product of their retrieval probabilities ($p_{ij} = p(e_i|e_h) \times p(e_j|e_t)$). We keep the rest of our model the same as the OWA setting.

Results. We compare the predictive performance of different models in terms of F1 score under each relation prediction task. As shown in Table 2, our model obtains very competitive performance compared with a comprehensive list of baseline methods. Specifically, on the prediction tasks of MAY_TREAT and CONTRAINDICATES, our model achieves a substantial improvement (1∼2 F1 score) and a very competitive performance on the task of SYMPTOM_OF and MAY_PREVENT. The small amount of training data might partly explain why our model does not perform so well in the CAUSES task. Such comparison shows the effectiveness of predicting relations based on associations and their relational interactions. Moreover, compared with those baseline models which encode graph structure into latent vector representation, our model utilizes co-occurrence graph more explicitly by leveraging the associative contexts symbolically to generate human-understandable rationales, which can assist medical experts as we will see shortly. In addition, we observe that our model consistently outperforms the CWA setting: Despite the CWA rationales are true statements on their own, they tend to have a low coverage of possible rationales, and thus, may be not so relevant for the target relation prediction, which leads to a poor predictive performance.

	OWA Rationales	CWA Rationales
Ranking Score	17	5
Avg. Sum Score/Case	6.14	2.24
Avg. Max Score/Case	2.04	0.77

Table 3: Human evaluation on the quality of rationales.

4.3 Model Rationale Evaluation

To measure the quality of our model rationales (i.e., OWA rationales), as well as to conduct an ablation study of our model, we conduct an expert evaluation for the OWA rationales and also compare them with the CWA rationales. We first collaborate with a physician to explore how much a model's rationales help them better trust the model's prediction following recent work for evaluating model interpretability (Singh et al., 2019; Mullenbach et al., 2018; Atutxa et al., 2019; Jin et al., 2020). Then, we present some case studies to show what kind of rationales our model has learnt. Note that compared with evaluation by human annotators for open-domain tasks (without expertise requirement), evaluation by medical experts is more challenging in general. The physician in our study (an M.D. with 9 years of clinical experience and currently a fellow trained in clinical informatics), who is able to understand the context of terms and the basics of the compared algorithms and can dedicate time, is qualified for our evaluation.

Expert Evaluation. We first explained to the

physician about the recall and recognition process in our framework and how model rationales are developed. They endorsed such reasoning process as one possible way to gain their trust in the model. Next, for each target pair for which our model correctly makes the prediction, they were shown the top-5 rationales produced by our framework and were asked whether each rationale helps them better trust the model prediction. For each rationale, they were asked to score it from 0 to 3 in which 0 is *no helpful*, 1 is *a little helpful*, 2 is *helpful* and 3 is *very helpful*. In addition to the individual rationale evaluation, we further compare the overall quality of CWA and OWA rationales, by letting experts rank them based the helpfulness of each set of rationales (the rationale set ranked higher gets 1 ranking score and both get 0 if they have the same rank). We refer readers to the appendix for more details of the evaluation protocol. We randomly select 30 cases in the MAY_TREAT relation and the overall evaluation results are summarized in Table 3. Out of 30, OWA wins in 17 cases and gets higher scores on individual rationales per case on average. There are 8 cases where the two sets of rationales are ranked the same[8] and 5 cases where CWA is better. To get a better idea of how the OWA model obtains more trust, we calculate the average sum score per case, which shows the OWA model gets a higher overall score per case. Considering in some cases only a few rationales are able to get non-zero scores, we also calculate the average max score per case, which shows that our OWA model generally provides one *helpful* rationale (score>2) per case. Overall, as we can see, the OWA rationales are more helpful to gain expert trust.

Case Study. Table 4 shows two concrete examples demonstrating what kind of model rationales our framework bases its predictions on. We highlight the rationales that receive high scores from the physician for being especially useful for trusting the prediction. As we can see, our framework is able to make correct predictions based on reasonable rationales. For instance, to predict that "cephalosporine" may treat "bacterial infection", our model relies on the rationale that "cefuroxime" may treat "infectious diseases". We also note that not all rationales are clinically established facts or even make sense, due to the unsupervised rationale learning and the probabilistic assumption formation process, which leaves space for future work to further improve the quality of rationales. Nevertheless, such model rationales can provide valuable information or new insights for clinicians. For another example, as pointed out by the physician, different medications possibly having the same treatment response, as shown in Case 2, could be clinically useful. That is, if three medications are predicted to possibly treat the same condition and a physician is only aware of two doing so, one might get insights into trying the third one. To summarize, our model is able to provide reasonable rationales and help users understand how model predictions are made in general.

[8]Of which, 7 cases are indicated equally unhelpful.

Case 1		
cephalosporins	may_treat	bacterial infection
cefuroxime	may_treat	viral syndrome
cefuroxime	may_treat	low grade fever
cefuroxime	**may_treat**	**infectious diseases**
cefuroxime	may_prevent	low grade fever
sulbactam	may_treat	low grade fever
Case 2		
azelastine	may_treat	perennial allergic rhinitis
astepro	**may_treat**	**perennial allergic rhinitis**
pseudoephedrine	**may_treat**	**perennial allergic rhinitis**
ciclesonide	**may_treat**	**perennial allergic rhinitis**
overbite	may_treat	perennial allergic rhinitis
diclofenac	may_treat	perennial allergic rhinitis

Table 4: Case studies for rationalizing medical relation prediction. For each case, the first panel is target pair and the second is top-5 rationales (**Bold** ones are useful rationales with high scores from the physician). The left (right) most column is the head (tail) term and their relational associations.

5 Related Work

Relation Extraction (RE) typically focuses on predicting relations between two entities based on their text mentions, and has been well studied in both open domain (Mintz et al., 2009; Zeng et al., 2015; Riedel et al., 2013; Lin et al., 2016; Song et al., 2019; Deng and Sun, 2019) and biomedical domain (Uzuner et al., 2011; Wang and Fan, 2014; Sahu et al., 2016; Lv et al., 2016; He et al., 2019). Among them, most state-of-the-art work develops various powerful neural models by leveraging human annotations, linguistic patterns, distance supervision, etc. More recently, an increasing amount of work has been proposed to improve model's transparency and interpretability. For example, Lee et al. (2019) visualizes self-attention weights learned from BERT (Devlin et al., 2019) to explain relation prediction. However, such text-based interpretable models tend to provide explanations within a *local*

context (e.g., words in a single sentence mentioning target entities), which may not capture a *holistic view* of all entities and their relations stored in a text corpus. We believe that such a holistic view is important for interpreting relations and can be provided to some degree by the *global statistics* from a text corpus. Moreover, global statistics have been widely used in the clinical domain as they can better preserve patient privacy (Finlayson et al., 2014; Wang et al., 2019).

On the other hand, in recent years, graph embedding techniques (Perozzi et al., 2014; Tang et al., 2015; Grover and Leskovec, 2016; Yue et al., 2019) have been widely applied to learn node representations based on graph structure. Representation learning based on global statistics from a text corpus (i.e., co-occurrence graph) has also been studied (Levy and Goldberg, 2014; Pennington et al., 2014). After employing such methods to learn entity embeddings, a number of relation classifiers (Nickel et al., 2011; Bordes et al., 2013; Socher et al., 2013; Yang et al., 2015; Wang et al., 2018) can be adopted for relation prediction. We compare our method with such frameworks to show its competitive predictive accuracy. However, such frameworks tend to be difficult to interpret as they provide little or no explanations on how decisions are made. In this paper, we focus more on model interpretability than predictive accuracy, and draw inspirations from existing cognitive theories of recall and recognition to develop a new framework, which is our core contribution.

Another line of research related to interpreting relation prediction is path-based knowledge graph (KG) reasoning (Gardner et al., 2014; Neelakantan et al., 2015; Guu et al., 2015; Xiong et al., 2017; Stadelmaier and Padó, 2019). In particular, existing paths mined from millions of relational links in knowledge graphs can be used to provide justifications for relation predictions. For example, to explain *Microsoft* and *USA* may hold the relation *CountryOfHeadquarters*, by traversing a KG, one can extract the path *Microsoft* $\xrightarrow{\text{IsBasedIn}}$ *Seattle* $\xrightarrow{\text{CountryLocatedIn}}$ *USA* as one explanation. However, such path-finding methods typically require large-scale relational links to infer path patterns, and cannot be applied to our co-occurrence graph as the co-occurrence links are unlabeled.

In addition, our work is closely related to the area of rationalizing machine decision by generating justifications/rationales accounting for model's prediction. In some scenarios, human rationales are provided as extra supervision for more explainable models (Zaidan et al., 2007; Bao et al., 2018). However, due to the high cost of manual annotation, model rationales are desired to be learned in an unsupervised manner(Lei et al., 2016; Bouchacourt and Denoyer, 2019; Zhao et al., 2019). For example, Lei et al. (2016) select a subset of words as rationales and Bouchacourt and Denoyer (2019) provide an explanation based on the absence or presence of "concepts", where the selected words and "concepts" are learned unsupervisedly. Different from text-based tasks, in this paper, we propose to rationalize relation prediction based on global co-occurrence statistics and similarly, model rationales in our work are captured without explicit manual annotation either, via a joint training framework.

6 Conclusion

In this paper, we propose an interpretable framework to rationalize medical relation prediction based on corpus-level statistics. Our framework is inspired by existing cognitive theories on human memory recall and recognition, and can be easily understood by users as well as provide reasonable explanations to justify its prediction. Essentially, it leverages corpus-level statistics to recall associative contexts and recognizes their relational connections as model rationales. Compared with a comprehensive list of baseline models, our model obtains competitive predictive performances. Moreover, we demonstrate its interpretability via expert evaluation and case studies.

Acknowledgments

We thank Srinivasan Parthasarathy, Ping Zhang, Samuel Yang and Kaushik Mani for valuable discussions. We also thank the anonymous reviewers for their hard work and constructive feedback. This research was sponsored in part by the Patient-Centered Outcomes Research Institute Funding ME-2017C1-6413, the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, and Ohio Supercomputer Center (Center, 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S.Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Accountability Act. 1996. Health insurance portability and accountability act of 1996. *Public law*, 104:191.

Aitziber Atutxa, Arantza Díaz de Ilarraza, Koldo Gojenola, Maite Oronoz, and Olatz Perez-de Viñaspre. 2019. Interpretable deep learning to map diagnostic texts to icd-10 codes. *International Journal of Medical Informatics*, 129:49–59.

Yujia Bao, Shiyu Chang, Mo Yu, and Regina Barzilay. 2018. Deriving machine attention from human rationales. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1903–1913.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems 26*, pages 2787–2795.

Diane Bouchacourt and Ludovic Denoyer. 2019. Educe: Explaining model decisions through unsupervised concepts extraction. *arXiv preprint arXiv:1905.11852*.

Raluca Budiu. 2014. Memory recognition and recall in user interfaces. *Nielsen Norman Group*.

Ohio Supercomputer Center. 1987. Ohio supercomputer center.

Ismail Ilkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. 2016. Open-world probabilistic databases. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. 2017. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 132–141.

Xiang Deng and Huan Sun. 2019. Leveraging 2-hop distant supervision from table entity pairs for relation extraction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 410–420.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Samuel G Finlayson, Paea LePendu, and Nigam H Shah. 2014. Building the graph of medicine from millions of clinical narratives. *Scientific data*, 1:140032.

Matt Gardner, Partha Talukdar, Jayant Krishnamurthy, and Tom Mitchell. 2014. Incorporating vector space similarity in random walk inference over knowledge bases. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 397–406.

Gary Gillund and Richard M Shiffrin. 1984. A retrieval model for both recognition and recall. *Psychological review*, 91(1):1–67.

Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 16, page 855864.

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing knowledge graphs in vector space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 318–327.

Bin He, Yi Guan, and Rui Dai. 2019. Classifying medical relations in clinical text via convolutional neural networks. *Artificial Intelligence in Medicine*, 93:43–49.

Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2010. SemEval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 33–38.

Xisen Jin, Zhongyu Wei, Junyi Du, Xiangyang Xue, and Xiang Ren. 2020. Towards hierarchical importance attribution: Explaining compositional semantics for neural sequence models. In *International Conference on Learning Representations*.

Rudolf Kadlec, Ondrej Bajgar, and Jan Kleindienst. 2017. Knowledge base completion: Baselines strike back. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 69–74.

Michael Kahana, Marc Howard, and Sean Polyn. 2008. Associative retrieval processes in episodic memory. *Psychology*.

D. P. Kingma and J. Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*.

Joohong Lee, Sangwoo Seo, and Yong Suk Choi. 2019. Semantic relation classification via bidirectional lstm networks with entity-aware attention using latent entity typing. *Symmetry*, 11(6):785.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 107–117.

Paea LePendu, Srinivasan V Iyer, Cédrick Fairon, and Nigam H Shah. 2012. Annotation analysis for testing drug safety signals using unstructured clinical notes. In *Journal of biomedical semantics*, volume 3, page S5. BioMed Central.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems 27*, pages 2177–2185.

Yankai Lin, Shiqi Shen, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. 2016. Neural relation extraction with selective attention over instances. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2124–2133.

Ryan Lowe, Nissan Pow, Iulian Serban, Laurent Charlin, and Joelle Pineau. 2015. Incorporating unstructured textual knowledge sources into neural dialogue systems. In *Neural information processing systems workshop on machine learning for spoken language understanding*.

Kevin Lund and Curt Burgess. 1996. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior research methods, instruments, & computers*, 28(2):203–208.

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*, pages 4765–4774.

Xinbo Lv, Yi Guan, Jinfeng Yang, and Jiawei Wu. 2016. Clinical relation extraction with deep learning. *International Journal of Hybrid Information Technology*, 9(7):237–248.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38.

Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011.

James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. 2018. Explainable prediction of medical codes from clinical text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1101–1111.

W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. 2019. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. 2015. Compositional vector space models for knowledge base completion. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 156–166.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML11, page 809816.

Peter A Nobel and Richard M Shiffrin. 2001. Retrieval processes in recognition and cued recall. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 27(2):384.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, et al. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 14, page 701710.

Meng Qu, Xiang Ren, Yu Zhang, and Jiawei Han. 2018. Weakly-supervised relation extraction by pattern-enhanced embedding learning. In *Proceedings of the 2018 World Wide Web Conference*, WWW 18, page 12571266.

Raymond Reiter. 1981. On closed world data bases. In *Readings in artificial intelligence*, pages 119–140. Elsevier.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 16, page 11351144.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling relations and their mentions without labeled text. In *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part III*, ECML PKDD10, page 148163.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. 2013. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 74–84.

Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215.

Sunil Sahu, Ashish Anand, Krishnadev Oruganty, and Mahanandeeshwar Gattu. 2016. Relation extraction from clinical texts using domain invariant convolutional neural network. In *Proceedings of the 15th Workshop on Biomedical Natural Language Processing*, pages 206–215.

Chandan Singh, W. James Murdoch, and Bin Yu. 2019. Hierarchical interpretations for neural network predictions. In *International Conference on Learning Representations*.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934.

Linfeng Song, Yue Zhang, Daniel Gildea, Mo Yu, Zhiguo Wang, and Jinsong Su. 2019. Leveraging dependency forest for neural medical relation extraction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 208–218.

Donald P Spence and Kimberly C Owens. 1990. Lexical co-occurrence and association strength. *Journal of Psycholinguistic Research*, 19(5):317–330.

Josua Stadelmaier and Sebastian Padó. 2019. Modeling paths for explainable knowledge base completion. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 147–157.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW 15, page 10671077.

Kristina Toutanova and Danqi Chen. 2015. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66.

Özlem Uzuner, Brett R South, Shuying Shen, and Scott L DuVall. 2011. 2010 i2b2/va challenge on concepts, assertions, and relations in clinical text. *Journal of the American Medical Informatics Association*, 18(5):552–556.

Shikhar Vashishth, Shyam Upadhyay, Gaurav Singh Tomar, and Manaal Faruqui. 2019. Attention interpretability across nlp tasks. *arXiv preprint arXiv:1909.11218*.

Patrick Verga, Emma Strubell, and Andrew McCallum. 2018. Simultaneously self-attending to all mentions for full-abstract biological relation extraction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 872–884.

Chang Wang, Liangliang Cao, and James Fan. 2016. Building joint spaces for relation extraction. In *IJCAI*, pages 2936–2942.

Chang Wang and James Fan. 2014. Medical relation extraction with manifold models. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 828–838.

Yanjie Wang, Rainer Gemulla, and Hui Li. 2018. On multi-relational link prediction with bilinear models. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zhen Wang, Xiang Yue, Soheil Moosavinasab, Yungui Huang, Simon Lin, and Huan Sun. 2019. Surfcon: Synonym discovery on privacy-aware clinical data. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD 19, page 15781586.

Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. DeepPath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 564–573.

Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question answering on Freebase via relation extraction and textual evidence. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2326–2336.

Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In

Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1480–1489.

Yuan Yao, Deming Ye, Peng Li, Xu Han, Yankai Lin, Zhenghao Liu, Zhiyuan Liu, Lixin Huang, Jie Zhou, and Maosong Sun. 2019. DocRED: A large-scale document-level relation extraction dataset. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 764–777.

Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. 2019. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, 36(4):1241–1251.

Omar Zaidan, Jason Eisner, and Christine Piatko. 2007. Using "annotator rationales" to improve machine learning for text categorization. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 260–267.

Daojian Zeng, Kang Liu, Yubo Chen, and Jun Zhao. 2015. Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1753–1762.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation classification via convolutional deep neural network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2335–2344.

Jie Zhao, Ziyu Guan, and Huan Sun. 2019. Riker: Mining rich keyword representations for interpretable product question answering. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD 19, page 13891398.

Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 207–212.

A Appendices

A.1 Implementation Details.

We implemented our model in Pytorch (Paszke et al., 2017) and optimized it by the Adam optimizer (Kingma and Ba, 2015). The dimension of term/node embeddings is set at 128. The number of negative triples for the relational learning is set at 100. The number of association contexts to use for assumption formation is 32. Early stopping is used when the performance in the dev set does not increase continuously for 10 epochs. We augment the relation triples for optimizing $\mathcal{L}_r$ (Eqn. 12) by adding their reverse relations for better training. We obtain DeepWalk and LINE (2nd) embeddings by OpenNE[9] and word2vec embeddings by doing SVD decomposition over the shifted PPMI co-occurrence matrix (Levy and Goldberg, 2014). Code, datasets, and more implementation details are available online[10].

A.2 Training Algorithm

Algorithm 1 CogStage Training Algorithm

INPUT: Corpus Statistics $\mathcal{G}$, Gold Triples $\mathcal{P}$, Binary Relation Data $\{(h_k, t_k), y_k\}_{k=1}^{M}$

OUTPUT: Model parameters

repeat
  Sample $\{e_i\}_{i=1}^{b_1}$ with gold contexts from $\mathcal{G}$
  for $i \leftarrow 1 : b_1$ **do**
    Calculate $p(e_j|e_i)$ and $\hat{p}(e_j|e_i)$
    Optimize $\mathcal{L}_n$ by Eqn. 3
  Sample $\{(h_i, r_i, t_i)\}_{i=1}^{b_2}$ from $\mathcal{P}$
  for $i \leftarrow 1 : b_2$ **do**
    Generate N_n corrupted triples
    Optimize $\mathcal{L}_r$ by Eqn. 12
  Sample $\{(h_i, t_i), y_i\}_{i=1}^{b_3}$
  for $i \leftarrow 1 : b_3$ **do**
    Calculate $p(e_j|h_i)$ and $p(e_j|t_i)$
    Get contexts $\{a_h^m\}_{m=1}^{N_c}$ and $\{a_t^n\}_{n=1}^{N_c}$
    Optimize $\mathcal{L}_p$ by Eqn. 14
until Convergence

[9] https://github.com/thunlp/OpenNE

[10] https://github.com/zhenwang9102/X-MedRELA

Evaluation Interface (Example)

All models predict the **may_treat** relation between t1 term unfractionated heparin ['unfractionated heparin [epc]', 'heparin'] and t2 term myocardial infarction (mi) ['myocardial infarction'] with the following rationales.

Please answer the following questions:

1. Are you familiar with t1 and t2 terms?

   ○ Yes ○ No ○ Kind of

2. Check each rationale and answer this question: Is which degree is rationale helpful for you to trust the prediction?

   (0: no helpful; 1: a little bit helpful; 2: helpful; 3: very helpful)

Model A's Rationale Set:

T1's contexts	Relational Interaction	T2's contexts	Score
metabolic alkalosis	may_prevent	myocardial infarction (mi)	
metabolic alkalosis	may_prevent	venous thrombosis	
rbbb	may_treat	myocardial infarction (mi)	
ards	symptom_of	myocardial infarction (mi)	
micronutrient	may_prevent	venous thrombosis	

Model B's Rationale Set:

T1's contexts	Relational Interaction	T2's contexts	Score
cardiac dysrhythmias	contraindicates	theophylline	
malignant neoplasm without specification of site	has_symptom	family history of cancer	
Iddm	contraindicates	glyburide	
morphine sulfate	contraindicated_by	respiratory depression	
insulin dependent diabetes	contraindicates	glyburide	

3. Please rank all sets of rationales based on overall how much they help you trust the model prediction (e.g., A > B). Note that it is ok to reject them if both models are unhelpful (A = B = 0).

Figure 4: Evaluation interface for expert evaluation.

Relations	UMLS Relations
May_treat	may_treat
May_prevent	may_prevent
Contraindicates	has_contraindicated_drug
Causes	cause_of; induces; causative_agent_of
Symptom of	disease_has_finding; disease_may_have_finding; has_associated_finding; has_manifestation; associated_condition_of; defining_characteristic_of

Table 5: Relations in our dataset and their mapped UMLS semantic relations. (UMLS relation "Treats" does not exist in our dataset and hence is not mapped with the "May_treat" relation.)

ReasonBERT: Pre-trained to Reason with Distant Supervision

Xiang Deng*[1], Yu Su[1], Alyssa Lees[2], You Wu[2], Cong Yu[2], and Huan Sun*[1]

[1]The Ohio State University, Columbus, OH
{deng.595,su.809,sun.397}@osu.edu
[2]Google Research, New York, NY
{alyssalees,wuyou,congyu}@google.com

Abstract

We present ReasonBERT, a pre-training method that augments language models with the ability to reason over long-range relations and multiple, possibly hybrid, contexts. Unlike existing pre-training methods that only harvest learning signals from local contexts of naturally occurring texts, we propose a generalized notion of distant supervision to automatically connect multiple pieces of text and tables to create pre-training examples that require long-range reasoning. Different types of reasoning are simulated, including intersecting multiple pieces of evidence, bridging from one piece of evidence to another, and detecting unanswerable cases. We conduct a comprehensive evaluation on a variety of extractive question answering datasets ranging from single-hop to multi-hop and from text-only to table-only to hybrid that require various reasoning capabilities and show that ReasonBERT achieves remarkable improvement over an array of strong baselines. Few-shot experiments further demonstrate that our pre-training method substantially improves sample efficiency.[1]

1 Introduction

Recent advances in pre-trained language models (LMs) have remarkably transformed the landscape of natural language processing. Pre-trained with self-supervised objectives such as autoregressive language modeling (Radford and Narasimhan, 2018; Radford et al., 2019; Brown et al., 2020) and masked language modeling (MLM) (Devlin et al., 2019; Liu et al., 2019b; Joshi et al., 2020), LMs encode a great deal of knowledge about language and significantly boost model performance on a wide range of downstream tasks (Liu et al., 2019a; Wang et al., 2019a,b) ranging from spell checking (Awasthi et al., 2019) to sentiment analysis (Xu et al., 2019) and semantic parsing (Rongali et al., 2020), just to name a few.

Existing self-supervised objectives for LM pre-training primarily focus on consecutive, naturally occurring text. For example, MLM enables LMs to correctly predict the missing word "*daughters*" in the sentence "*Obama has two __ , Malia and Sasha.*" based on the local context and the knowledge stored in the parameters. However, many tasks require reasoning beyond local contexts: multi-hop question answering (QA) (Yang et al., 2018; Welbl et al., 2018) and fact verification (Jiang et al., 2020) require reasoning over multiple pieces of evidence, hybrid QA (Chen et al., 2020) requires simultaneously reasoning over unstructured text and structured tables, and dialogue systems require reasoning over the whole dialogue history to accurately understand the current user utterance (Andreas et al., 2020).

To address this limitation in existing LM pre-training, we propose ReasonBERT, a pre-training method to augment LMs for explicitly reasoning over long-range relations and multiple contexts. ReasonBERT pairs a query sentence with multiple relevant pieces of evidence drawn from possibly different places and defines a new LM pre-training objective, *span reasoning*, to recover entity spans that are masked out from the query sentence by jointly reasoning over the query sentence and the relevant evidence (Figure 1). In addition to text, we also include tables as evidence to further empower LMs to reason over hybrid contexts.

One major challenge in developing ReasonBERT lies in how to create a large set of query-evidence pairs for pre-training. Unlike existing self-supervised pre-training methods, examples with complex reasoning cannot be easily harvested from naturally occurring texts. Instead, we draw inspiration from *distant supervision* (Mintz et al., 2009a), which assumes

*Corresponding authors.

[1]Our code and pre-trained models are available at https://github.com/sunlab-osu/ReasonBERT.

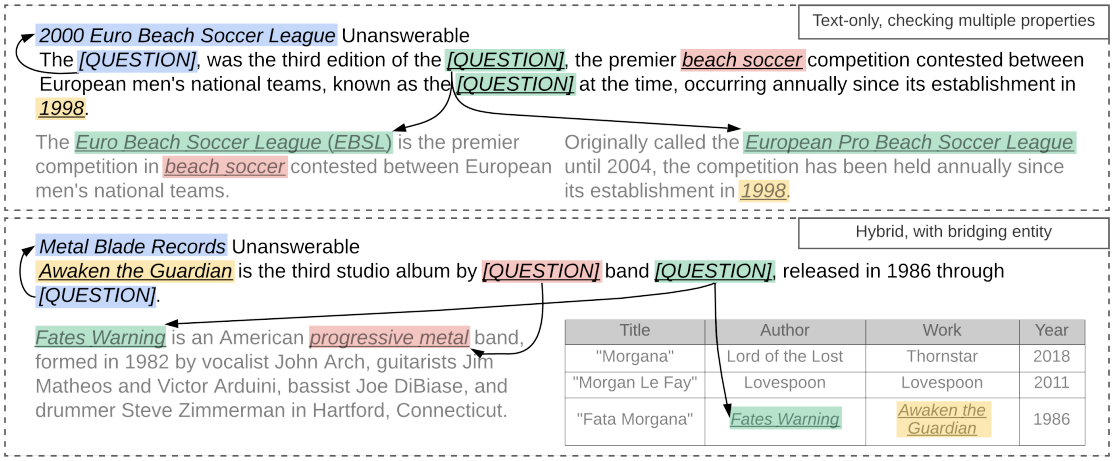


Figure 1: Examples of our pre-training data acquired via distant supervision, which covers a wide range of topics with both textual and tabular evidence. For each query sentence (in black), we first select two pairs of entities (underlined) to find two pieces of evidence (in grey) via distant supervision. We then randomly mask one entity from each selected pair and aim to recover it by reasoning over the evidence. Note that the two selected pairs may share a common entity; in case this entity is masked, we can mimic different types of multi-hop reasoning, e.g., intersection (Ex. 1) and bridging (Ex. 2). To simulate unanswerable cases, we additionally mask one entity (in blue) that does not exist in the evidence. Figure best viewed in color.

that "*any sentence containing a pair of entities that are known to participate in a relation is likely to express that relation*," and generalize it to our setting of multiple pieces of evidence from text and tables. Specifically, given a query sentence containing an entity pair, if we mask one of the entities, another sentence or table that contains the same pair of entities can likely be used as evidence to recover the masked entity. Moreover, to encourage deeper reasoning, we collect multiple pieces of evidence that are jointly used to recover the masked entities in the query sentence, allowing us to scatter the masked entities among different pieces of evidence to mimic different types of reasoning. Figure 1 illustrates several examples using such distant supervision. In Ex. 1, a model needs to check multiple constraints (i.e., intersection reasoning type) and find "*the beach soccer competition that is established in 1998*." In Ex. 2, a model needs to find "*the type of the band that released Awaken the Guardian*," by first inferring the name of the band "*Fates Warning*" (i.e., bridging reasoning type).

We first replace the masked entities in a query sentence with the `[QUESTION]` tokens. The new pre-training objective, span reasoning, then extracts the masked entities from the provided evidence. We augment existing LMs like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019b) by continuing to train them with the new objective, which leads to ReasonBERT, a new LM with better reasoning capabilities. Then query sentence and textual evidence are encoded via the LM. When tabular evidence is present, we use the structure-aware transformer TAPAS (Herzig et al., 2020) as the encoder to capture the table structure.

We evaluate ReasonBERT on the extractive QA task, which is arguably the most representative task requiring reasoning about world knowledge. We conduct a comprehensive evaluation using a variety of popular datasets: MRQA (Fisch et al., 2019), a single-hop QA benchmark including six datasets from different domains; HotpotQA (Yang et al., 2018), a multi-hop QA dataset; NQTables, a subset of the Natural Questions dataset (Kwiatkowski et al., 2019) where answers can be found in tables; and HybridQA (Chen et al., 2020), a hybrid multi-hop QA dataset that requires reasoning over both tables and text. Under the few-shot setting, ReasonBERT substantially outperforms the baselines in almost all datasets, demonstrating that the reasoning ability learned from pre-training can easily transfer to downstream QA tasks and generalize well across domains. Under the full-data setting, ReasonBERT obtains substantial gains in multi-hop and hybrid QA datasets. Despite its simple model architecture, ReasonBERT achieves similar or better performance compared with more sophisticated state-of-the-art models for each dataset.

2 Background

Language model pre-training. Existing pre-training objectives such as MLMs (Devlin et al., 2019; Joshi et al., 2020) tend to implicitly memorize the learned knowledge in the parameters of the underlying neural network. In this work, we aim to augment pre-training by encouraging a model to *reason* about (instead of memorizing) world knowledge over the given contexts.

Extractive question answering. To measure a model's reasoning ability about world knowledge, we select extractive QA as a downstream task, which is perhaps one of the most representative tasks for this purpose. Given a question q and provided evidence E, an extractive QA model $p_\theta(a|q, E)$ aims to select a contiguous span a from E that answers the question, or output a special token if E is not sufficient to answer the question.

Our approach, ReasonBERT, is inspired by this formulation and extends it to language model pre-training. The challenge in defining such a self-supervised task is in the creation of question-evidence pairs from unlabeled data. Moreover, we aim for a generic approach that works for a wide range of extractive QA settings including single-hop and multi-hop reasoning, hybrid contexts with both unstructured texts and structured tables, as well as few-shot settings. We discuss how to address the challenge and achieve this goal in the next two sections.

3 Distant Supervision (DS) for Pre-training

We use English Wikipedia as our data source for pre-training. We first extract sentences and tables from Wikipedia pages and then identify salient spans (such as named entities) from them. We apply the idea of distant supervision and match the sentences and tables to form query-evidence pairs, which are used to create pre-training examples.

3.1 Data Collection

Text. We first extract paragraphs from Wikipedia pages and split them into sentences. We consider named entities including both real-world entities (e.g., person, location) and temporal and numeric expressions (e.g., date and quantity) as potential answer entities for pre-training. We first identify real-world entities using existing hyperlinks. Since Wikipedia pages generally do not contain links to themselves, we additionally detect such self-mentions by searching the names and aliases of the topic entity for each page. Temporal and numeric expressions are identified using an existing NER tool[2].

Tables. We extract tables that are labeled as <wikitable> from Wikipedia, and only consider tables with no more than 500 cells. First, real-world entities are detected using existing hyperlinks. Unlike our method employed for textual sentences, we do not use traditional NER tools here as they are not tailored to work well on tables. Instead, for a cell that does not contain hyperlinks, we match the complete cell value with sentences that are closely related to the table, sourced either from the same page or a page containing a hyperlink pointing to the current page. If the matched span in the sentence contains a named entity, we consider the same entity as being linked to the cell as well. Otherwise we consider this cell as a unique entity in the table.

Please see Appendix A.1 for details about the tools and resources we use.

3.2 Query-Evidence Pairing via DS

As described in Section 2, a standard QA sample is composed of a question, an answer and evidence. The model infers the relationship between the answer and other entities in the question, and extracts it from the evidence. In this work, we try to simulate such samples in pre-training. Given a sentence with entities, it can be viewed as a question by masking some entities as answers for prediction. The key issue is then how to find evidence that contains not only the answer entity, but also the relational information for inference. Here we borrow the idea of distant supervision (Mintz et al., 2009b).

Given a sentence as a query, we first extract pairs of entities in it. For each entity pair, we then find other sentences and tables that also contain the same pair as evidence. Since we do not have the known relation constraint in the original assumption of distant supervision, we use the following heuristics to collect evidence that has high quality relational knowledge about the entities and is relevant to the query. First, we only consider entity pairs that contain at least one real-world entity. For textual evidence, the entity pair needs to contain the topic entity of the Wikipedia page, which is more likely to have relations to other entities. For

[2] https://nlp.johnsnowlabs.com/

Setting	# queries	# sent.	# tab.	# ent. pairs
Text-only	7.6M	8.4M	-	5.5M
Hybrid	3.2M	4.3M	0.9M	6.0M

Table 1: Statistics about the pre-training data.

Setting	All the same	One different	All different
Text-only	50%	30%	20%
Hybrid	60%	8%	32%

Table 2: Analysis of pre-training data quality with 50 examples for each setting. *One different* is when the relation between the selected entities is different from the relation expressed in the query sentence for of the two pieces of evidence.

tabular evidence, we consider only entity pairs that are in the same row of the table, but they do not need to contain the topic entity, as in many cases the topic entity is not present in the tables. In both cases, the query and evidence should come from the same page, or the query contains a hyperlink pointing to the evidence page. For tabular evidence, we also allow for the case where the table contains a hyperlink pointing to the query page.

3.3 Pre-training Data Generation

Given the query-evidence pairs, a naive way to construct pre-training examples is to sample a single piece of evidence for the query, and mask a shared entity as "answer", as in Glass et al. (2020). However, this only simulates simple single-hop questions. In this work, we construct complex pre-training examples that require the model to conduct multi-hop reasoning. Here we draw inspiration from how people constructed multi-hop QA datasets. Take HotpotQA (Yang et al., 2018) as an example. It first collected candidate evidence pairs that contain two paragraphs (A, B), with a hyperlink from A to B so that the topic entity of B is a bridging entity that connects A and B. Crowd workers then wrote questions based on each evidence pair. Inspired by this process, we combine multiple pieces of evidence in each pre-training example and predict multiple masked entities simultaneously. The detailed process is described below. Figure 1 shows two examples. For more examples, please check Appendix A.1.

We start by sampling up to two entity pairs from the query sentence and one piece of evidence (sentence or table) for each entity pair. We then mask one entity in each pair as the "answer" to predict. The resulting pre-training examples fall into three categories: (1) Two disjoint entity pairs $\{(a, b), (c, d)\}$ are sampled from the query, and one entity from each pair, e.g., $\{a, c\}$, is masked. This is similar to a combination of two single-hop questions. (2) The two sampled entity pairs $\{(a, b), (b, c)\}$ share a common entity b, and b is masked. The model needs to find two sets of entities that respectively satisfy the relationship with a and c, and take an intersection (Type II in HotpotQA; see Ex. 1 in Figure 1). (3) The two sampled entity pairs $\{(a, b), (b, c)\}$ share a common entity b, and $\{b, c\}$ are masked. Here b is the bridging entity that connects a and c. The model needs to first identify b and then recover c based on its relationship with b (Type I and Type III in HotpotQA; see Ex. 2 in Figure 1). We also mask an entity from the query that is not shown in the evidence to simulate unanswerable cases. All sampling is done randomly during pre-training.

3.4 Data Statistics and Analysis

We prepare pre-training data for two settings: (1) one with only textual evidence (text-only) and (2) the other including at least one piece of tabular evidence in each sample (hybrid). Some statistics of the collected data are summarized in Table 1. For the text-only setting, we extract approximately 7.6M query sentences, each containing 2 entity pairs that are matched with 3 different pieces of textual evidence on average. For the hybrid setting, we select approximately 3.2M query sentences, each containing 3.5 entity pairs, matched with 5.8 different pieces of evidence on average.

We also conduct an analysis of the pre-training data quality using 50 randomly sampled examples from each setting. We compare the query sentence and the evidence to see if they are expressing the same relation between the selected entities. Results are summarized in Table 2. We can see that in both settings, almost 70% of the examples have the desired characteristic that the evidence contains useful relational knowledge for recovering missing entities in the query sentence.

4 Pre-training

4.1 Encoder

For the text-only setting, we use the standard transformer encoder in BERT (Devlin et al., 2019). For settings where the input contains tables, we adopt the transformer variant recently introduced in TAPAS (Herzig et al., 2020), which uses extra token-type embeddings (indicating the row/column

position of a token) to model the table structure.

4.2 Span Reasoning Objective

Now we describe our *span reasoning objective*, which can advance the reasoning capabilities of a pre-trained model.

Given a sample collected for pre-training as described in Section 3.3, we replace the masked entities $\mathcal{A} = \{a_1, \ldots, a_n\}$ ($n \leq 3$) in the query sentence q with special `[QUESTION]` tokens. The task then becomes recovering these masked entities from the given evidence E (concatenation of the sampled evidence). Specifically, we first concatenate q, E and add special tokens to form the input sequence as $[\texttt{[CLS]}, q, \texttt{[SEP]}, E]$, and get the contextualized representation $\mathbf{x}$ with the encoder. Since we have multiple entities in q masked with `[QUESTION]`, for each a_i, we use its associated `[QUESTION]` representation as a dynamic query vector $\mathbf{x}_{a_i}$ to extract its start and end position s, e of a_i in E (i.e., *question-aware* answer extraction).

$$
\begin{aligned}
P(s|q,E) &= \frac{\exp\left(\mathbf{x}_s^\top \mathbf{S}\mathbf{x}_{a_i}\right)}{\sum_k \exp\left(\mathbf{x}_k^\top \mathbf{S}\mathbf{x}_{a_i}\right)} \\
P(e|q,E) &= \frac{\exp\left(\mathbf{x}_e^\top \mathbf{E}\mathbf{x}_{a_i}\right)}{\sum_k \exp\left(\mathbf{x}_k^\top \mathbf{E}\mathbf{x}_{a_i}\right)}
\end{aligned} \tag{1}
$$

Here $\mathbf{S}, \mathbf{E}$ are trainable parameters. $\mathbf{x}_{a_i}$ is the representation of special token `[QUESTION]` corresponding to a_i; $\mathbf{x}_k$ is the representation of the k-th token in E. If no answer can be found in the provided evidence, we set s, e to point to the `[CLS]` token.

The *span reasoning* loss is then calculated as follows:

$$
L_{SR} = -\sum_{a_i \in \mathcal{A}} \left(\log P\left(s_{a_i}|q,E\right) + \log P\left(e_{a_i}|q,E\right)\right) \tag{2}
$$

We name this objective as *span reasoning*, as it differs from the *span prediction/selection* objectives in existing pre-training work such as SpanBert (Joshi et al., 2020), Splinter (Ram et al., 2021), and SSPT (Glass et al., 2020) in the following ways: (1) Unlike SpanBert and Splinter that use single contiguous paragraph as context, where the models may focus on local cues, we encourage the model to do long-range contextualization by including both query and evidence as input, which can come from different passages, and recovering the masked entities by grounding them on the evidence E. (2) Unlike SSPT, we improve the model's ability to reason across multiple pieces of evidence by including two disjoint pieces of evidence in a single sample and scattering the answer entities among them to mimic different types of reasoning chains. (3) We mimic the scenario where a span cannot be inferred based on the given contexts, by masking entities in q that do not appear in E, in which case the model is trained to select the special `[CLS]` token.

	MRQA	HotpotQA	NQTables	HybridQA
# train	86136.5	88881	17112	62686
# dev	-	1566	1901	3466
# test	9704	7405	1118	3463
# evidence	1	10	8.7	34.7
# tokens*	374.9	89.1	289.6	156.3
has text/table	✓/✗	✓/✗	✗/✓	✓/✓

Table 3: Dataset statistics. The statistics for MRQA are averaged over all 6 datasets. # tokens* is the average number of tokens per evidence.

4.3 Final Objective

We also include the *masked language modeling* (MLM) objective in pre-training to leverage other tokens in the input that are not entities. In particular, we randomly mask tokens that are not an entity or token in the header row for tables, and use an MLM objective to recover them. Following the default parameters from BERT, we use a masking probability of 15%.

The final loss is the sum of *span reasoning* loss and *masked language modeling* loss. Following previous work (Glass et al., 2020; Herzig et al., 2020), we initialize with a pre-trained encoder, and extend the pre-training with our objectives. For the text part, we pre-train two models with BERT-Base (denoted as ReasonBERT$_{\text{B}}$) and RoBERTa-Base (denoted as ReasonBERT$_{\text{R}}$); for the table part, we use TAPAS-Base (denoted as ReasonBERT$_{\text{T}}$). More implementation details of pre-training are included in Appendix A.2.

5 Experiments

5.1 Datasets

We conduct experiments with a wide range of extractive QA datasets. Statistics are summarized in Table 3.

MRQA (Fisch et al., 2019). A single-hop extractive QA benchmark that unifies various existing QA datasets into the same format. Here we use the in-domain subset that contains 6 datasets: SQuAD (Rajpurkar et al., 2016), NewsQA (Trischler et al., 2017), TriviaQA (Joshi et al., 2017), SearchQA (Dunn et al., 2017), HotpotQA (Yang et al., 2018) and Natural Questions (Kwiatkowski et al., 2019). Similar to Ram et al. (2021), we adapt these

Train. Size	Model	SQuAD	TriviaQA	NQ	NewsQA	SearchQA	HotpotQA	Average
16	BERT	9.9±0.6	15.4±1.3	20.5±1.5	6.5±1.2	16.8±1.2	9.6±1.6	13.1
	RoBERTa	10.3±1.1	21.0±3.1	22.5±2.1	6.7±2.0	23.4±3.5	11.2±1.0	15.9
	SpanBERT	15.7±3.6	27.4±4.1	24.3±2.1	8.1±1.4	24.1±3.2	16.3±2.0	19.3
	SSPT	10.8±1.2	21.2±3.8	23.7±4.1	6.5±1.9	25.8±2.6	9.1±1.5	16.2
	Splinter	16.7±5.9	23.9±3.8	25.1±2.8	11.6±1.0	23.6±4.5	15.1±3.5	19.3
	Splinter*	**54.6**	18.9	27.4	**20.8**	26.3	24.0	28.7
	ReasonBERT$_B$	33.2±4.0	37.2±2.6	33.1±2.7	11.8±2.3	**46.1**±5.2	22.4±2.8	30.6
	ReasonBERT$_R$	41.3±5.5	**45.5**±5.8	**33.6**±3.9	16.2±3.2	45.8±4.5	**34.1**±2.9	**36.1**
128	BERT	21.5±1.4	23.9±0.8	31.7±0.8	11.3±1.3	32.6±2.3	14.0±0.8	22.5
	RoBERTa	48.8±4.2	36.0±2.9	36.4±2.0	22.8±2.4	41.3±2.0	35.2±1.4	36.7
	SpanBERT	61.2±4.7	48.8±6.6	38.8±2.6	31.0±5.3	50.0±3.7	44.0±2.3	45.7
	SSPT	41.5±5.0	30.3±3.7	35.0±2.4	14.0±3.6	42.8±3.5	23.7±3.4	31.2
	Splinter	55.0±10.3	45.7±4.1	41.1±2.7	33.9±2.8	48.8±3.7	46.9±7.1	45.2
	Splinter*	**72.7**	44.7	46.3	**43.5**	47.2	54.7	51.5
	ReasonBERT$_B$	58.5±2.2	56.2±0.6	46.7±2.6	27.8±0.6	60.8±1.7	45.2±2.3	49.2
	ReasonBERT$_R$	66.7±2.9	**62.1**±0.9	**49.8**±1.6	35.7±1.5	**62.3**±1.7	**57.2**±0.6	**55.6**
1024	BERT	64.1±0.9	41.6±2.6	50.1±0.6	43.0±0.3	53.1±1.0	46.5±1.9	49.7
	RoBERTa	77.9±0.5	62.2±1.3	60.3±0.6	55.0±0.5	67.5±0.8	63.4±0.8	64.4
	SpanBERT	81.1±0.7	67.0±1.0	63.2±0.9	56.4±0.4	70.0±0.8	67.6±1.1	67.5
	SSPT	77.6±1.4	60.1±2.0	58.7±0.7	52.8±1.1	65.9±0.8	63.3±1.6	63.1
	Splinter	79.8±3.5	67.3±1.5	63.8±0.5	54.6±1.4	68.9±0.3	68.4±1.2	67.1
	Splinter*	**82.8**	64.8	**65.5**	**57.3**	67.3	**70.3**	68.0
	ReasonBERT$_B$	76.9±0.5	67.4±0.5	63.6±0.6	52.2±0.5	70.6±0.6	67.8±0.5	66.4
	ReasonBERT$_R$	79.7±0.3	**70.1**±0.2	65.0±0.9	54.7±0.6	**72.8**±0.4	69.7±0.6	**68.7**
All	BERT	88.8	73.6	78.7	67.5	82.0	76.2	77.8
	RoBERTa	92.0	78.1	80.6	**71.9**	85.2	79.1	81.2
	SpanBERT	**92.5**	**79.9**	80.7	71.1	84.8	**80.7**	**81.6**
	SSPT	91.1	77.0	80.0	69.7	83.3	79.7	80.1
	Splinter	92.4	79.7	80.3	70.8	84.0	80.6	81.3
	Splinter*	92.2	76.5	**81.0**	71.3	83.0	**80.7**	80.8
	ReasonBERT$_B$	90.3	77.5	79.9	68.7	83.7	80.5	80.1
	ReasonBERT$_R$	91.4	78.9	80.8	71.4	**85.3**	80.6	81.4

Table 4: Results on MRQA datasets. **Best** and Second Best results are highlighted. We report the average F1 score over five runs for each dataset, and the macro-average of the six datasets. Splinter* is the result reported in the original paper, where the authors use a deeper model with additional transformation layers on top of the encoder.

datasets to the few-shot setting by randomly sampling smaller subsets from the original training set for training, and use the original development set for testing.

HotpotQA (Yang et al., 2018). A multi-hop QA dataset that requires reasoning over multiple pieces of evidence. Here we follow the distractor setting, where 10 paragraphs are provided to answer a question while only two of them contain relevant information. We split 10% of the original train-hard split for development, and use the original development set for testing.

NQTables (Kwiatkowski et al., 2019). A subset of the Natural Questions dataset, where at least one answer to the question is present in a table. We extract 19,013 examples from the original training set (307,373 examples) and split them with a 9:1 ratio for training and development. The test set is then created from the original development split (7,830 examples) and contains 1,118 examples. Here we only keep tables from the original Wikipedia article as evidence. Similar subsets are also used in Herzig et al. (2021) and Zayats et al. (2021).

HybridQA (Chen et al., 2020). A multi-hop QA dataset with hybrid contexts. Each example contains a table and several linked paragraphs.

We adopt the evaluation script from MRQA[3], which evaluates the predicted answer using exact match (EM) and token-level F1 metrics.

5.2 Baselines

BERT (Devlin et al., 2019). A deep transformer model pre-trained with masked languge model (MLM) and next sentence prediction objectives.

RoBERTa (Liu et al., 2019b). An optimized version of BERT that is pre-trained with a larger text

[3] https://github.com/mrqa/MRQA-Shared-Task-2019

corpus.

SpanBERT (Joshi et al., 2020). A pre-training method designed to better represent and predict spans of text. It extends BERT by masking contiguous random spans, and training the span boundary representation to predict the entire masked span.

SSPT (Glass et al., 2020). A pre-training method designed to improve question answering by training on cloze-like training instances. Unlike ReasonBERT, SSPT only masks a single span in the query sentence and predicts it based on an evidence paragraph provided by a separate retriever.

Splinter (Ram et al., 2021). A pre-training method optimized for few-shot question answering, where the model is pre-trained by masking and predicting recurring spans in a passage.

TAPAS (Herzig et al., 2020). A pre-training method designed to learn representations for tables. The model is pre-trained with MLM on tables and surrounding texts extracted from Wikipedia.

For fair comparison, in each task, we use the same model architecture with different pre-trained encoders, which is similar to the one used for span reasoning in pre-training. We append the `[QUESTION]` token to a question and construct the input sequence the same way as in pre-training. We then score all the start, end locations and rank all spans (s, e) (See Eqn. 3 and 4 in Appendix). We use a pre-trained encoder and learn the answer extraction layers ($\mathbf{S}$, $\mathbf{E}$ in Eqn. 1) from scratch during fine-tuning.

Unless otherwise stated, we use the pre-trained base version so that all models have similar capacity (110M parameters for ReasonBERT$_{\text{B}}$, 125M parameters for ReasonBERT$_{\text{R}}$, and 111M parameters for ReasonBERT$_{\text{T}}$).

5.3 Few-shot Single-hop Text QA

We first experiment with the easier, single-hop MRQA benchmark under the few-shot setting to show that our pre-training approach learns general knowledge that can be transferred to downstream QA tasks effectively. Results are shown in Table 4. We can see that ReasonBERT outperforms pre-trained language models such as BERT, RoBERTa and SpanBERT by a large margin on all datasets, particularly with an average absolute gain of 20.3% and 14.5% over BERT and RoBERTa respectively. Compared with pre-training methods such as SSPT and Splinter, ReasonBERT also shows superior performance and obtains the best results on average.

Model	Recall		1%		Full	
	Top 2	Top 3	F1	EM	F1	EM
HGN$_{\text{RoBERTa-Large}}$	-	-	-	-	82.2	-
HGN$_{\text{BERT}}$	-	-	-	-	74.8	-
BERT	92.4	96.9	39.8	28.6	71.9	57.9
RoBERTa	93.1	97.5	56.0	43.1	76.3	62.9
SpanBERT	93.6	97.7	56.5	44.1	76.3	62.9
SSPT	93.9	97.9	54.7	41.8	75.4	61.5
Splinter	94.1	97.9	57.0	44.2	76.5	62.5
ReasonBERT$_{\text{B}}$	93.8	97.8	57.6	45.3	77.2	63.4
ReasonBERT$_{\text{R}}$	**94.0**	**98.0**	**63.1**	**50.2**	**78.1**	**64.8**

Table 5: Results on HotpotQA.

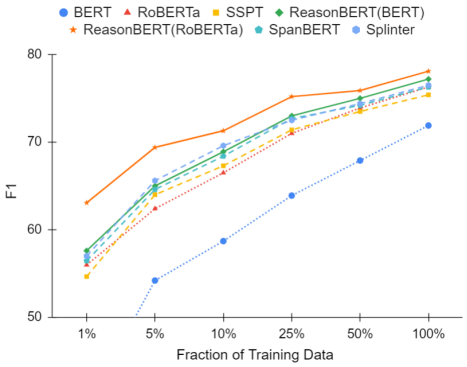


Figure 2: Few-shot learning results on HotpotQA.

Under the full-data setting, ReasonBERT performs competitively and all methods achieve similarly high accuracy. We still demonstrate improvements upon BERT and RoBERTa, and ReasonBERT$_{\text{R}}$ second best average score.

5.4 Multi-hop Text QA

To demonstrate that our approach is useful in conducting deep reasoning over multiple contexts, we experiment with the HotpotQA dataset. Here we design a simplified multi-hop QA model that first selects relevant paragraphs as evidence, and then extracts the answer from the top selected evidence. Please see Appendix A.3 for implementation details. In addition to comparing ReasonBERT with other pre-training methods using the same base model, we also show results for HGN (Fang et al., 2020), which is one of the top ranked models on the HotpotQA leaderboard that uses a more sophisticated model design.

Results are shown in Table 5. All models perform very well for evidence selection, with over 96% top 3 recall, but ReasonBERT still maintains a slim lead over baselines. ReasonBERT provides a 5.3% improvement for BERT and a 1.8% improvement for RoBERTa on overall F1 score, and outperforms all other pre-training methods. ReasonBERT also outperforms the HGN model with BERT, but

Model	Dev		Test	
	F1	EM	F1	EM
RoBERTa	58.9	52.8	63.6	58.1
ReasonBERT$_R$	61.9	56.4	66.3	60.9
TAPAS	64.9	57.8	65.9	59.6
ReasonBERT$_T$	**69.2**	**63.5**	**72.5**	**67.3**

Table 6: Results on NQTables.

is lower than the one using RoBERTa-Large, which is probably due to simpler design and smaller size of the model. We further experiment under the few-shot setting. Here we focus on the QA performance, so we reuse the evidence selector trained with full data for each model, and train the QA module with different fractions of training data. We can see that the advantage of using ReasonBERT is more obvious with limited training data. With 1% of training data, ReasonBERT$_R$ obtains F1 score of 63.1%, a 7.1% absolute gain over RoBERTa. Results for training the QA model with different fraction of training data is shown in Figure 2. We can see that ReasonBERT obtains larger gain under the few-shot setting.

5.5 Table QA

We demonstrate our approach also works with structured data such as tables using the NQTables dataset. We first use a text based RoBERTa encoder as baseline, which linearizes a table as a text sequence, by concatenating tokens row by row and separating cells with the `[SEP]` token. We then experiment with the structure-aware encoder from TAPAS and compare the pre-trained TAPAS encoder with the one pre-trained using ReasonBERT. Results are shown in Table 6. First, we can see that TAPAS outperforms RoBERTa by 2.3%, demonstrating the importance of modeling the table structure. ReasonBERT$_R$ slightly outperforms TAPAS on test set, but ReasonBERT$_T$ further boosts F1 to 72.5%, resulting in at least 6.6% absolute gains over existing methods. Results for training the Table QA model with different fractions of training data are shown in Figure 3. ReasonBERT$_T$ consistently outperforms TAPAS while ReasonBERT$_R$ gradually matches the performance of TAPAS with the increasing of training data.

5.6 Hybrid QA

We further evaluate our approach on HybridQA, a multi-hop question answering dataset using both text and tables as evidence. Chen et al. (2020) proposes a baseline model HYBRIDER that divides

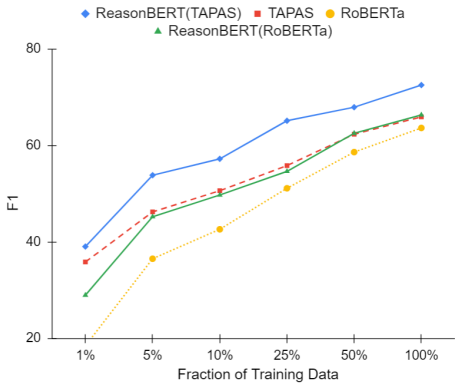


Figure 3: Few-shot learning results on NQTables.

Model	Cell Selection		Dev		Test	
	Top 1	Top 2	F1	EM	F1	EM
HYBRIDER$_{\text{BERT-Base}}$	-	-	50.9	43.7	50.2	42.5
HYBRIDER$_{\text{BERT-Large}}$	68.5	-	50.7	44.0	50.6	43.8
TAPAS+RoBERTa	73.3	79.7	64.0	57.3	63.3	56.1
ReasonBERT	**76.1**	**81.3**	**67.2**	**60.3**	**65.3**	**58.0**

Table 7: Results on HybridQA.

the problem into four tasks: linking, ranking, hopping and reading comprehension. We follow their design but simplify the model by merging ranking and hopping into a single cell selection task. We use the linking results from Chen et al. (2020), and then train a table based cell selector to select the cell which is the answer or is linked to the passage that contains the answer. Finally, we train a text based QA model to extract the final answer by taking the table snippet that contains the selected cell, and concatenating it with the hyperlinked passage as evidence. Please see Appendix A.3 for implementation details. Results are shown in Table 7. First, we can see that our simplified architecture works surprisingly well, with TAPAS for cell selection and RoBERTa for QA, we already outperform HYBRIDER. The performance is further improved by replacing the encoders with ReasonBERT$_T$ and ReasonBERT$_R$, and substantially outperforms the best model on the leaderboard (52.04 EM) at the time of submission.

6 Ablation Study

We further conduct ablation studies on HotpotQA to verify our design choices, summarized in Table 8. Here we remove different components of ReasonBERT$_R$ and test them under both the full-data and few-shot setting (with 1024 examples). To save computing resources, here all models are pre-trained with 5 epochs. We can see that combining multiple pieces of evidence and predicting

Model	1024		Full	
	F1	EM	F1	EM
ReasonBERT$_R$	65.2	52.8	79.2	65.8
– MLM	63.7	51.3	77.7	64.0
– Unanswerable Ent.	64.4	51.8	78.4	65.0
– Multiple Evidences	60.8	48.6	77.8	64.5

Table 8: Ablation study on HotpotQA.

multiple masked spans simultaneously brings the most gain, especially under the few-shot setting. This is probably because the setting allows us to simulate complex reasoning chains and encourage the model to do deep reasoning. Masking unanswerable entities and utilizing MLM also help to improve performance.

7 Related Work

Language Model Pre-training. Contextualized word representations pre-trained on large-scale unlabeled text corpus have been widely used in NLP lately. Most prevalent approaches are variants of pre-trained language models such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019b). More recently, self-supervised pre-training has also shown promising results on modalities other than plain text, such as tables (Herzig et al., 2020; Deng et al., 2020; Iida et al., 2021), knowledge bases (Zhang et al., 2019; Peters et al., 2019) and image-text (Su et al., 2020). Meanwhile, there has also been work that uses pre-training to accommodate specific needs of downstream NLP tasks, such as open-domain retrieval (Guu et al., 2020), representing and predicting spans of text (Joshi et al., 2020) and semantic parsing (Yu et al., 2020; Deng et al., 2021).

Machine Reading Comprehension. Machine reading comprehension (MRC) or extractive QA has become an important testbed for natural language understanding evaluation (Fisch et al., 2019). The conventional method to train an MRC model usually relies on large-scale supervised training data (Chen et al., 2017; Zhang et al., 2020). Recently, more and more work has focused on developing self-supervised methods that can reduce the need for labeled data for more efficient domain adaptation, while achieving the same or even better performance. One direction is question generation (Pan et al., 2021), which automatically generates questions and answers from unstructured and structured data sources using rules or neural generators. Recent work also tries to directly simulate questions with cloze-like query sentences. Splinter (Ram et al., 2021) proposes to pre-train the model by masking and predicting recurring spans. However, this limits the query and context to come from the same passage. In contrast, SSPT (Glass et al., 2020) also pre-trains with a span selection objective, but uses a separate document retriever to get relevant paragraphs as context.

Our work is most related to SSPT, but uses distant supervision to collect query-evidence pairs and thus obviate the need for a retriever. Meanwhile, to encourage the model to learn complex reasoning, we mimic different types of reasoning chains by masking multiple entities, including unanswerable ones, and simultaneously inferring them from disjoint pieces of evidence. Our method also works with heterogeneous sources including both text and tables, while most existing work considers only text-based question answering.

8 Conclusion and Future Work

We propose ReasonBERT, a novel pre-training method to enhance the reasoning ability of language models. The resulting model obtains substantial improvements on multi-hop and hybrid QA tasks that require complex reasoning, and demonstrates superior few-shot performance. In the future, we plan to use our query-evidence pairs collected by distant supervision to improve the retrieval performance for open-domain QA, as well as empower ReasonBERT to handle more types of reasoning, like comparison and numeric reasoning, in natural language understanding.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. Authors at The Ohio State University were sponsored in part by Google Faculty Award, the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, NSF CAREER #1942980, Fujitsu gift grant, and Ohio Supercomputer Center (Center, 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein. Research was also supported with Cloud TPUs from Google's TPU Research Cloud (TRC).

References

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 2019. Parallel iterative edit models for local sequence transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4260–4270, Hong Kong, China. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ohio Supercomputer Center. 1987. Ohio supercomputer center.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350, Online. Association for Computational Linguistics.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. Turl: table understanding through representation learning. *Proceedings of the VLDB Endowment*, 14(3):307–319.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Matthew Dunn, Levent Sagun, Mike Higgins, V. U. Güney, Volkan Cirik, and Kyunghyun Cho. 2017. Searchqa: A new q&a dataset augmented with context from a search engine. *ArXiv*, abs/1704.05179.

Yuwei Fang, Siqi Sun, Zhe Gan, Rohit Pillai, Shuohang Wang, and Jingjing Liu. 2020. Hierarchical graph network for multi-hop question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8823–8838, Online. Association for Computational Linguistics.

Adam Fisch, Alon Talmor, Robin Jia, Minjoon Seo, Eunsol Choi, and Danqi Chen. 2019. MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 1–13, Hong Kong, China. Association for Computational Linguistics.

Michael Glass, Alfio Gliozzo, Rishav Chakravarti, Anthony Ferritto, Lin Pan, G P Shrivatsa Bhargav, Dinesh Garg, and Avi Sil. 2020. Span selection pre-training for question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2773–2782, Online. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*.

Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Eisenschlos. 2021. Open domain question answering over tables via dense retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 512–519, Online. Association for Computational Linguistics.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained representations of tabular data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3446–3456, Online. Association for Computational Linguistics.

Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. 2020. HoVer: A dataset for many-hop fact extraction and claim verification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3441–3460, Online. Association for Computational Linguistics.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, Florence, Italy. Association for Computational Linguistics.

Y. Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009a. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011, Suntec, Singapore. Association for Computational Linguistics.

Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009b. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011, Suntec, Singapore. Association for Computational Linguistics.

Liangming Pan, Wenhu Chen, Wenhan Xiong, Min-Yen Kan, and William Yang Wang. 2021. Unsupervised multi-hop question answering by question generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5866–5880, Online. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54, Hong Kong, China. Association for Computational Linguistics.

Alec Radford and Karthik Narasimhan. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Ori Ram, Yuval Kirstain, Jonathan Berant, Amir Globerson, and Omer Levy. 2021. Few-shot question answering by pretraining span selection. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3066–3079, Online. Association for Computational Linguistics.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. Don't parse, generate! A sequence to sequence architecture for task-oriented semantic parsing. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 2962–2968. ACM / IW3C2.

Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. 2020. VL-BERT: pre-training of generic visual-linguistic representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200, Vancouver, Canada. Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. 2018. Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6:287–302.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Hu Xu, Bing Liu, Lei Shu, and Philip Yu. 2019. BERT post-training for review reading comprehension and aspect-based sentiment analysis. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2324–2335, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Caiming Xiong, et al. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

Vicky Zayats, Kristina Toutanova, and Mari Ostendorf. 2021. Representations for question answering from documents with tables and text. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2895–2906, Online. Association for Computational Linguistics.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451, Florence, Italy. Association for Computational Linguistics.

Zhuosheng Zhang, J. Yang, and Hai Zhao. 2020. Retrospective reader for machine reading comprehension. *ArXiv*, abs/2001.09694.

A Implementation Details

A.1 Pre-training Data Details

We extract paragraphs from Wikipedia XML dump[4] use JWPL[5] and tables use wikitextparser[6]. The paragraphs are then processed with SparkNLP[7] for sentence boundary detection and named entity recognition.

Table 9 and Table 10 show some examples of the query-evidence pairs we collected for pre-training. The selected entities are underlined. During pre-training, we will mask some of the entities in the query and recover them based on the evidence. As the pre-training data is collected via distant supervision, it contains some noise. Here we also include some bad examples where the evidence does not express the same relation between the selected entities as the query sentence (highlighted in red).

A.2 Pre-training Details

We set the max length of query sentences to 100 tokens and the max length of single piece of evidence to 200 if there are two evidence selections or 400 if there is only one. For textual evidence, we include the neighbouring sentences from the same paragraph as extra context for the selected evidence sentence and clip to the max evidence length. For tabular evidence, we take a snippet of the original table, and truncate the cells to 20 tokens. We always keep the first row and column in the table, as they often contain important information such as headers and subject entities. Based on the selected entity pair, we sample up to 5 columns and include as many rows as possible until reaching the budget.

We initialize our encoder with BERT-Base[8] and RoBERTa-Base[9] for the text part, and TAPAS-base[10] for the table part. We train ReasonBERT using AdamW (Loshchilov and Hutter, 2019) for 10 epochs with batches of 256 sequences of length 512; this is approximately 290k steps with text-only data, and 120k steps with hybrid data. We base our implementation on Huggingface Transformers (Wolf et al., 2020), and train on a single eight-core TPU on the Google Cloud Platform.

[4] https://dumps.wikimedia.org/

[5] https://dkpro.github.io/dkpro-jwpl/

[6] https://github.com/5j9/wikitextparser

[7] https://nlp.johnsnowlabs.com/

[8] https://huggingface.co/bert-base-uncased/tree/main

[9] https://huggingface.co/roberta-base/tree/main

[10] https://huggingface.co/google/tapas-base/tree/no_reset

A.3 Fine-tuning Details

To extract the answer span from given evidence, we score all the start, end locations and rank all spans (s, e) by $g(s, e|q, E)$ as follows:

$$f_{\mathbf{start}} = \mathbf{x}_s^\top \mathbf{S}\mathbf{x}_q, \quad f_{\mathbf{end}} = \mathbf{x}_e^\top \mathbf{E}\mathbf{x}_q \tag{3}$$

$$\begin{aligned} g(s, e|q, E) = & f_{\mathbf{start}}(s|q, E) \\ & + f_{\mathbf{end}}(e|q, E) \\ & - f_{\mathbf{start}}(\texttt{[CLS]}|q, E) \\ & - f_{\mathbf{end}}(\texttt{[CLS]}|q, E) \end{aligned} \tag{4}$$

For all fine-tuning experiments, we set the batch size to 20 and use a maximal learning rate of $5 \cdot 10^{-5}$, which warms up in the first 10% of the steps, and then decays linearly. We use the development set for model selection if it is present, otherwise we use the last model checkpoint.

Single-hop text QA. We split the text sequence to fit the max input length by sliding a window with a stride of 128 tokens.

For the few-shot setting, we fine-tune the model for either 10 epochs or 200 steps (whichever is larger). For the fully supervised setting, we fine-tune the model for 2 epochs.

Multi-hop text QA. We design a simplified multi-hop QA model that first selects relevant paragraphs as evidence, and then extracts the answer from the selected evidence samples. Specifically, we first generate all possible paragraphs by sliding a 200-token window over all articles with a stride of 128 tokens. We then train an evidence selector to pick the top 3 evidence samples. As the information for answering a question in HotpotQA is scattered in two articles, we list all possible combinations of paragraphs that come from two different articles and concatenate them together to form the final evidence. We then use the base QA model to extract the answer based on the question and the combined evidence.

We fine-tune the evidence selector model for 2 epochs, and the QA model for 5 epochs with full data. For the few-shot setting, we fine-tune the QA model for 10 epochs with 1&, 5% and 10% of the training data, and for 5 epochs with 25% and 50% of the training data.

Table QA. For the text based model, We split the text sequence to fit the max input length by sliding a window with a stride of 128 tokens. For the table based model, we truncate each cell to 50 tokens,

and split the table into snippets horizontally. Same as pre-training, we include the first row and column in each table snippet.

We fine-tune the model for 5 epochs with full data. For the few-shot setting, we fine-tune the QA model for 10 epochs with 1&, 5% and 10% of the training data, and for 5 epochs with 25% and 50% of the training data.

Hybrid QA. Chen et al. (2020) proposes a baseline model that divides the problem into four tasks: 1) linking: link questions to their corresponding cells using heuristics. 2) ranking: rank the linked cells use a neural model. 3) hopping: based on the cell selected in the last step, decide which neighboring cell or itself contains the final answer. 4) reading comprehension: extract the answer from the predicted cell or its linked paragraph. We follow their design and simplify the model by merging ranking and hopping into a single cell selection task. We use the linking results from Chen et al. (2020). For each linked cell, we take a snippet out of the original table including the headers, the entire row of the linked cell, and concatenate the evidence sentence to the cell if it is linked through the hyperlinked passage. To select the cell, we train the model to select separately on the token, row and column level, and aggregate the final scores . More specifically, we calculate the probability of selecting on the token and row level as follows:

$$
\begin{aligned}
P(t|q,E) &= \frac{\exp\left(\mathbf{x}_t^\top \mathbf{S}\mathbf{x}_{a_i}\right)}{\sum_k \exp\left(\mathbf{x}_k^\top \mathbf{S}\mathbf{x}_{a_i}\right)} \\
S_{cell} &= \text{mean}_{x_i \in cell}\left(\mathbf{x}_i^\top \mathbf{R}\mathbf{x}_a\right) \\
P(r_a = j \mid q,E) &= \frac{\exp\left(\max_{cell \in r_j} S_{cell}\right)}{\sum_k \exp\left(\max_{cell \in r_k} S_{cell}\right)}
\end{aligned}
\tag{5}
$$

Here $\mathbf{S}$ is the weight matrix of the token selection header, we only consider the first token in each cell, and t is the first token of the selected cell. $\mathbf{R}$ is the weight matrix of row selection header, and the column selection probability is calculated similarly with another column selection header. We first score each cell by averaging over all tokens in that cell. We then do a max pooling over all cells in the row or column so the model can focus on the strongest signal, for example the column header. The final probability of selecting a cell is the sum of token, row and column scores.

The input for the QA model then contains the header of the table, the row of the selected cell, and the hyperlinked passage.

We fine-tune the cell selection model for 2 epochs and the QA model for 3 epochs.

Query	Evidence
"*I Thought I Lost You*" was nominated for Broadcast Film Critics Association Award for Best Song and Golden Globe Award for Best Original Song, but lost both to *Bruce Springsteen*'s "*The Wrestler*" from *The Wrestler* (2008).	On January 11, 2009, *Springsteen* won the Golden Globe Award for Best Song for "*The Wrestler*", from the Darren Aronofsky *film by the same name*. "*I Thought I Lost You*" was nominated for the Broadcast Film Critics Association Award for Best Song at the 14th Broadcast Film Critics Association Award, but lost to *Bruce Springsteen*'s "*The Wrestler*" from *The Wrestler* (2008).
Film critic *Roger Ebert* compared it to *John Carpenter*'s *Halloween*, noting: "Blue Steel" is a sophisticated update of Halloween, the movie that first made *Jamie Lee Curtis* a star.	Historian Nicholas Rogers notes that film critics contend that Carpenter's direction and camera work made *Halloween* a "resounding success." *Roger Ebert* remarks, ... Since *Jamie Lee Curtis*, the main actress from the original and the sequel Halloween II (1981), wanted to reunite the cast and crew of the original film, she asked *Carpenter* to direct Halloween H20: 20 Years Later.
A hybrid disc is an optical disc that has multiple *file system* installed on it, typically *ISO 9660* and HFS+ (or *HFS* on older discs).	Hierarchical File System (*HFS*) is a proprietary *file system* developed by Apple Inc. for use in computer systems running Mac OS. *ISO 9660* is a *file system* for optical disc media. Being sold by the International Organization for Standardization (ISO) the file system is considered an international technical standard.
After *1709* , the heads of the *House of Orleans* branch of *the House of Bourbon* ranked as the prince of the Blood – this meant that the dukes could be addressed as Monsieur le Prince (a style they did not, however, use).	From *1709* until the French Revolution, the *Orleans* dukes were next in the order of succession to the French throne after members of the senior branch of the House of Bourbon, descended from Louis XIV. Restored briefly in 1814 and definitively in 1815 after the fall of the First French Empire, the senior line of the Bourbons was finally overthrown in the July Revolution of 1830. A cadet *Bourbon* branch, the *House of Orleans*, then ruled for 18 years (1830–1848), until it too was overthrown.
The *Citroen C6* is an *executive car* produced by the French car maker *Citroen* from 2005 to *2012*.	The *C6* was aimed as a stylish alternative to *executive cars*, like the BMW 5 Series and the Audi A6, and it has been described as "spaceship that rides on air", "charmingly idiosyncratic" and "refreshingly different". In *2012*, *Citroen* announced plans to enter the World Touring Car Championship. The team transformed a DS3 WRC into a laboratory vehicle to help with early development, while ...
Leaving the *Market Street subway* at Ferry Portal heading south, the T Third Street follows *The Embarcadero* south of Market Street, then veers onto King Street in front of *Oracle Park* until it reaches the *Caltrain* station terminal.	the 4th & King *Caltrain* station is 1.5 blocks from the stadium, and the *Oracle Park* Ferry Terminal is outside the east edge of the ballpark beyond the center field bleachers. the southwestern end of the *Market Street subway* connects to the much-older Twin Peaks Tunnel, and the northeastern end connects to surface tracks along the *The Embarcadero*.

Table 9: Pre-training data examples, text-only setting.

<table>
<tr><th>Query</th><th>Evidence</th></tr>
<tr><td>Rowland Barran (7 August 1858 – 6 August 1949) was an English Liberal Party politician and Member of Parliament.</td><td>Rowland Barran was the youngest son of Sir John Barran, a pioneer in clothing manufacture and Member of Parliament for Leeds and Otley.<table><tr><th>Year</th><th>Member</th><th>Party</th></tr><tr><td>1885</td><td>William Jackson</td><td>Conservative</td></tr><tr><td>1902</td><td>Rowland Barran</td><td>Liberal</td></tr><tr><td>1918</td><td>Alexander Farquharson</td><td>Coalition Liberal</td></tr></table></td></tr>
<tr><td>"It Ain't Over 'til It's Over" is a song recorded, written, and produced by American musician Lenny Kravitz for his second studio album, Mama Said (1991).</td><td>Retrieved on August 19, 2007. Kravitz's biggest single yet, "It Ain't Over 'til It's Over", went to number 2 on the Billboard Hot 100.<table><tr><th>Act</th><th>Order</th><th>Song</th><th>Rock Artist</th></tr><tr><td>In Stereo</td><td>4</td><td>Demons</td><td>Imagine Dragons</td></tr><tr><td>Cyrus Villanueva</td><td>5</td><td>It Ain't Over 'til It's Over</td><td>Lenny Kravitz</td></tr><tr><td>Michaela Baranov</td><td>6</td><td>Wild Horses</td><td>The Rolling Stones</td></tr></table></td></tr>
<tr><td>Ronnie Bremer raced the first five races of the season with Brooks Associates Racing, before moving to Polestar Motor Racing.</td><td><table><tr><th>Place</th><th>Name</th><th>Team</th></tr><tr><td>5</td><td>Ronnie Bremer</td><td>Brooks Associates Racing</td></tr><tr><td>6</td><td>Bryan Sellers</td><td>Lynx Racing</td></tr><tr><td>9</td><td>Jonathan Bomarito</td><td>Transnet Racing</td></tr></table></td></tr>
<tr><td>Donovan also appeared in the 1980 film Breaker Morant, but in a subsidiary role, rather than as the title character.</td><td><table><tr><th>Title</th><th>Year</th><th>Role</th></tr><tr><td>Cop Shop (TV series)</td><td>1978-1980</td><td>Detective Sgt. Vic</td></tr><tr><td>Breaker Morant</td><td>1980</td><td>Captain Simon Hunt</td></tr><tr><td>Smash Palace</td><td>1981</td><td>Traffic Officer</td></tr></table></td></tr>
<tr><td>Try a Little Kindness is the sixteenth album by American singer/guitarist Glen Campbell, released in 1970.</td><td>At the height of his popularity, a 1970 biography by Freda Kramer, The Glen Campbell Story, was published.<table><tr><th>Day</th><th>Album</th><th>Artist</th><th>Notes</th></tr><tr><td>1</td><td>On the Boards</td><td>Taste</td><td>-</td></tr><tr><td>26</td><td>Chicago</td><td>Chicago</td><td>aka Chicago II</td></tr><tr><td>-</td><td>Try a Little Kindness</td><td>Glen Campbell</td><td>-</td></tr></table></td></tr>
</table>

Table 10: Pre-training data examples, hybrid setting.

TURL: Table Understanding through Representation Learning

Xiang Deng*
The Ohio State University
Columbus, Ohio
deng.595@buckeyemail.osu.edu

Huan Sun*
The Ohio State University
Columbus, Ohio
sun.397@osu.edu

Alyssa Lees
Google Research
New York, NY
alyssalees@google.com

You Wu
Google Research
New York, NY
wuyou@google.com

Cong Yu
Google Research
New York, NY
congyu@google.com

ABSTRACT

Relational tables on the Web store a vast amount of knowledge. Owing to the wealth of such tables, there has been tremendous progress on a variety of tasks in the area of table understanding. However, existing work generally relies on heavily-engineered task-specific features and model architectures. In this paper, we present TURL, a novel framework that introduces the pre-training/fine-tuning paradigm to relational Web tables. During pre-training, our framework learns deep contextualized representations on relational tables in an unsupervised manner. Its universal model design with pre-trained representations can be applied to a wide range of tasks with minimal task-specific fine-tuning.

Specifically, we propose a structure-aware Transformer encoder to model the row-column structure of relational tables, and present a new Masked Entity Recovery (MER) objective for pre-training to capture the semantics and knowledge in large-scale unlabeled data. We systematically evaluate TURL with a benchmark consisting of 6 different tasks for table understanding (e.g., relation extraction, cell filling). We show that TURL generalizes well to all tasks and substantially outperforms existing methods in almost all instances.

PVLDB Reference Format:
Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: Table Understanding through Representation Learning. PVLDB, 14(3): 307 - 319, 2021.
doi:10.14778/3430915.3430921

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sunlab-osu/TURL.

1 INTRODUCTION

Relational tables are in abundance on the Web and store a large amount of knowledge, often with key entities in one column and attributes in the others. Over the past decade, various large-scale collections of such tables have been aggregated [4, 6, 7, 25]. For example, Cafarella et al. [6, 7] reported 154M relational tables out of a total of 14.1 billion tables in 2008. More recently, Bhagavatula et al. [4] extracted 1.6M high-quality relational tables from Wikipedia. Owing to the wealth and utility of these datasets, various tasks such as table interpretation [4, 16, 29, 35, 47, 48], table augmentation [1, 6, 12, 42, 45, 46], etc., have made tremendous progress in the past few years.

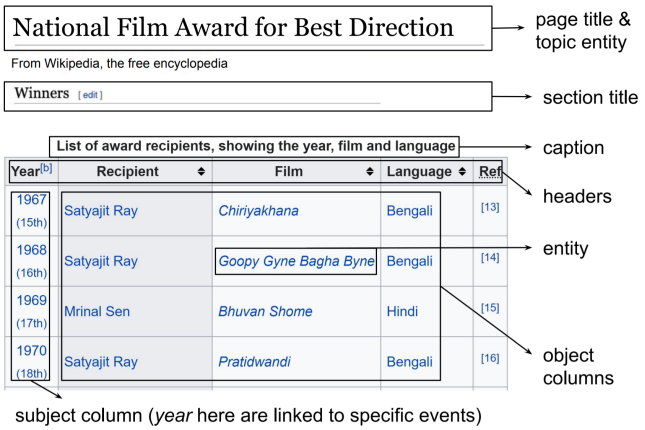


Figure 1: An example of a relational table from Wikipedia.

However, previous work such as [4, 45, 46] often rely on heavily-engineered task-specific methods such as simple statistical/language features or straightforward string matching. These techniques suffer from several disadvantages. First, simple features only capture shallow patterns and often fail to handle the flexible schema and varied expressions in Web tables. Second, task-specific features and model architectures require effort to design and do not generalize well across tasks.

Recently, the pre-training/fine-tuning paradigm has achieved notable success on unstructured text data. Advanced language models such as BERT [15] can be pre-trained on large-scale unsupervised text and subsequently fine-tuned on downstream tasks using task-specific supervision. In contrast, little effort has been extended to the study of such paradigms on relational tables. Our work fills this research gap.

Promising results in some table based tasks were achieved by the representation learning model of [13]. This work serializes a table into a sequence of words and entities (similar to text data) and learns embedding vectors for words and entities using Word2Vec [28]. However, [13] cannot generate *contextualized* representations, i.e., it does not consider varied use of words/entities in different contexts

*Corresponding authors.

Proceedings of the VLDB Endowment, Vol. 14, No. 3 ISSN 2150-8097.
doi:10.14778/3430915.3430921

and only produces a single fixed embedding vector for word/entity. In addition, *shallow* neural models like Word2Vec have relatively limited learning capabilities, which hinder the capture of complex semantic knowledge contained in relational tables.

We propose TURL, a novel framework for learning deep contextualized representations on relational tables via pre-training in an unsupervised manner and task-specific fine-tuning.

There are two main challenges in the development of TURL: (1) *Relational table encoding*. Existing neural network encoders are designed for linearized sequence input and are a good fit with unstructured texts. However, data in relational tables is organized in a semi-structured format. Moreover, a relational table contains multiple components including the table caption, headers and cell values. The challenge is to develop a means of modeling the row-and-column structure as well as integrating the heterogeneous information from different components of the table. (2) *Factual knowledge modeling*. Pre-trained language models like BERT [15] and ELMo [31] focus on modeling the syntactic and semantic characteristics of word use in natural sentences. However, relational tables contain a vast amount of factual knowledge about entities, which cannot be captured by existing language models directly. Effectively modelling such knowledge in TURL is a second challenge.

To address the first challenge, we encode information from different table components into separate input embeddings and fuse them together. We then employ a *structure-aware* Transformer [38] encoder with masked self-attention. The conventional Transformer model is a bi-directional encoder, so each element (i.e., token/entity) can attend to all other elements in the sequence. We explicitly model the row-and-column structure by restraining each element to only aggregate information from other structurally related elements. To achieve this, we build a visibility matrix based on the table structure and use it as an additional mask for the self-attention layer.

For the second challenge, we first learn embeddings for each entity during pre-training. We then model the relation between entities in the same row or column with the assistance of the visibility matrix. Finally, we propose a Masked Entity Recovery (MER) pre-training objective. The technique randomly masks out entities in a table with the objective of recovering the masked items based on other entities and the table context (e.g., caption/header). *This encourages the model to learn factual knowledge from the tables and encode it into entity embeddings.* In addition, we utilize the entity mention by keeping it as additional information for a certain percentage of masked entities. *This helps our model build connections between words and entities* . We also adopt the Masked Language Model (MLM) objective from BERT, which aims to model the complex characteristics of word use in table metadata.

We pre-train our model on 570K relational tables from Wikipedia to generate contextualized representations for tokens and entities in the relational tables. We then fine-tune our model for specific downstream tasks using task-specific labeled data. A distinguishing feature of TURL is its universal architecture across different tasks - only minimal modification is needed to cope with each downstream task. To facilitate research in this direction, we compiled a benchmark that consists of 6 diverse tasks, including entity linking, column type annotation, relation extraction, row population, cell filling and schema augmentation. We created new datasets in addition to including results for existing datasets when publicly available. Experimental results show that TURL substantially outperforms existing task-specific and shallow Word2Vec based methods.

Table 1: Summary of notations for our table data.

Symbol	Description
T	A relational table $T = (C, H, E, e_t)$
C	Table caption (a sequence of tokens)
H	Table schema $H = \{h_0, ..., h_i, ..., h_m\}$
h_i	A column header (a sequence of tokens)
E	Columns in table that contains entities
e_t	The topic entity of the table $e_t = (e_t^{\mathrm{e}}, e_t^{\mathrm{m}})$
e	An entity cell $e = (e^{\mathrm{e}}, e^{\mathrm{m}})$

Our contributions are summarized as follows:

- To the best of our knowledge, TURL is the first framework that introduces the pre-training/fine-tuning paradigm to relational Web tables. The pre-trained representations along with the universal model design save tremendous effort on engineering task-specific features and architectures.
- We propose a structure-aware Transformer encoder to model the structure information in relational tables. We also present a novel Masked Entity Recovery (MER) pre-training objective to learn the semantics as well as the factual knowledge about entities in relational tables.
- To facilitate research in this direction, we present a benchmark that consists of 6 different tasks for table interpretation and augmentation. We show that TURL generalizes well to various tasks and substantially outperforms existing models. Our source code, benchmark, as well as pre-trained models will be available online.

2 PRELIMINARY

We now present our data model and give a formal task definition:

In this work, we focus on relational Web tables and are most interested in the factual knowledge about entities. Each table $T \in \mathcal{T}$ is associated with the following: (1) Table caption C, which is a short text description summarizing what the table is about. When the page title or section title of a table is available, we concatenate these with the table caption. (2) Table headers H, which define the table schema; (3) Topic entity e_t, which describes what the table is about and is usually extracted from the table caption or page title; (4) Table cells E containing entities. Each entity cell $e \in E$ contains a specific object with a unique identifier. For each cell, we define the entity as $e = (e^{\mathrm{e}}, e^{\mathrm{m}})$, where e^{e} is the specific entity linked to the cell and e^{m} is the entity mention (i.e., the text string).

(C, H, e_t) is also known as *table metadata* and E is the actual *table content*. Notations used in the data model are summarized in Table 1.

Explicitly, we study the unsupervised representation learning on relational Web tables, which is defined as follows.

Definition 2.1. Given a relational Web table corpus, our representation learning task aims to learn in an unsupervised manner a task-agnostic contextualized vector representation for each token in all table captions C's and headers H's and for each entity (i.e., all entity cells E's and topic entities e_t's).

3 RELATED WORK

Representation Learning. The pre-training/fine-tuning paradigm has drawn tremendous attention in recent years. Extensive effort has been devoted to the development of unsupervised representation learning methods for both unstructured text and structured knowledge bases, which in turn can be utilized for a wide variety of downstream tasks via fine-tuning.

Earlier work, including Word2Vec [28] and GloVe [30], pre-train distributed representations for words on large collections of documents. The resulting representations are widely used as input embeddings and offer significant improvements over randomly initialized parameters. However, pre-trained word embeddings suffer from word polysemy: they cannot model varied word use across linguistic contexts. This complexity motivated the development of contextualized word representations [15, 31, 43] . Instead of learning fixed embeddings per word, these works construct language models that learn the joint probabilities of sentences. Such pre-trained language models have had huge success and yield state-of-the-art results on various NLP tasks [39].

Similarly, unsupervised representation learning has also been adopted in the space of structured data like knowledge bases (KB) and databases. Entities and relations in KB have been embedded into continuous vector spaces that still preserve the inherent structure of the KB [37]. These entity and relation embeddings are utilized by a variety of tasks, such as KB completion [5, 40], relation extraction [34, 41], entity resolution [18], etc. Similarly, [17] learned embeddings for heterogeneous data in databases and used it for data integration tasks.

More recently, there has been a corpus of work incorporating knowledge information into pre-trained language models [32, 49]. ERNIE [49] injects knowledge base information into a pre-trained BERT model by utilizing pre-trained KB embeddings and a denoising entity autoencoder objective. The experimental results demonstrate that knowledge information is extremely helpful for tasks such as entity linking, entity typing and relation extraction.

Despite the success of representation learning on text and KB, few works have thoroughly explored contextualized representation learning on relational Web tables. Pre-trained language models are directly adopted in [26] for entity matching. Two recent papers from the NLP community [20, 44] study pre-training on Web tables to assist in semantic parsing or question answering tasks on tables. In this work, we introduce TURL, a new methodology for learning deep contextualized representations for relational Web tables that preserve both semantic and knowledge information. In addition, we conduct comprehensive experiments on a much wider range of table-related tasks.

Table Interpretation. The Web stores large amounts of knowledge in relational tables. Table interpretation aims to uncover the semantic attributes of the data contained in relational tables, and transform this information into machine understandable knowledge. This task is usually accomplished with help from existing knowledge bases. In turn, the extracted knowledge can be used for KB construction and population.

There are three main tasks for table interpretation: entity linking, column type annotation and relation extraction [4, 47]. Entity linking is the task of detecting and disambiguating specific entities mentioned in a table. Since relational tables are centered around entities, entity linking is a key step for table interpretation, and a fundamental component to many table-related tasks [47]. [4] employed a graphical model, and used a collective classification technique to optimize a global coherence score for a set of entities in a table. [35] presented the T2K framework, which is an iterative matching approach that combines both schema and entity matching. More recently, [16] introduced a hybrid method that combines both entity lookup and entity embeddings, which resulted in superior performance on various benchmarks.

Column type annotation and relation extraction both work with table columns. The former aims to annotate columns with KB types while the latter intends to use KB predicates to interpret relations between column pairs. Prior work has generally coupled these two tasks with entity linking [29, 35, 48]. After linking cells to entities, the types and relations associated with the entities in KB can then be used to annotate columns. In recent work, column annotation without entity linking has been explored [10, 11, 21]. These works modify text classification models to fit relational tables and have shown promising results. Moreover, relation extraction on web tables has also been studied for KB augmentation [8, 14, 36].

Table Augmentation. Tables are a popular data format to organize and present relational information. Users often have to manually compose tables when gathering information. It is desirable to offer some intelligent assistance to the user, which motivates the study of table augmentation [45]. Table augmentation refers to the task of expanding a seed query table with additional data. Specifically, for relational tables this can be divided into three sub-tasks: row population for retrieving entities for the subject column [12, 45], cell filling that fills the cell values for given subject entities [1, 42, 46] and schema augmentation that recommends headers to complete the table schema [6, 45]. For row population tasks, [12] searches for complement tables that are semantically related to seed entities and the top ranked tables are used for population. [45] further incorporates knowledge base information with a table corpus, and develops a generative probabilistic model to rank candidate entities with entity similarity features. For cell filling, [42] uses the query table to search for matching tables, and extracts attribute values from those tables. More recently, [46] proposed the CellAutoComplete framework that makes use of a large table corpus and a knowledge base as data sources, and incorporates preprocessing, candidate value finding, and value ranking components. In terms of schema augmentation, [6] tackles this problem by utilizing an attribute correlation statistics database (ACSDb) collected from a table corpus. [45] utilizes a similar approach to the row population techniques and ranks candidate headers with sets of features.

Existing benchmarks. Several benchmarks have been proposed for table interpretation: (1) T2Dv2 [25] proposed in 2016 contains 779 tables from various websites. It contains 546 relational tables, with 25119 entity annotations, 237 table-to-class annotations and 618 attribute-to-property annotations. (2) Limaye et al. [27] proposed a benchmark in 2010 which contains 296 tables from Wikipedia. It was used in [16] for entity linking, and was also used in [11] for column type annotation. (3) Efthymiou et al. [16] created a benchmark (referred to as "WikiGS" in our experiments) that includes 485,096 tables from Wikipedia. WikiGS was originally

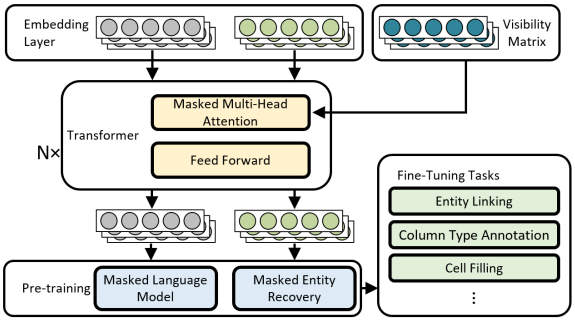


Figure 2: Overview of our TURL framework.

used for entity linking with 4,453,329 entity matches. [11] further annotated a subset of it containing 620 entity columns with 31 DBpedia types and used it for column type annotation. (4) The recent SemTab 2019 [23] challenge also aims at benchmarking systems that match tabular data to KBs, including three tasks, i.e., assigning a semantic type to an column, matching a cell to an entity, and assigning a property to the relationship between two columns. It used sampled tables from T2Dv2 [25] and WikiGS [16] in the first two rounds, and automatically generated tables in later rounds.

In contrast to table interpretation, few benchmarks have been released for table augmentation. Zhang et al. [45] studied row population and schema augmentation with 2000 randomly sampled Wikipedia tables in total for validation and testing. [46] curated a test collection with 200 columns containing 1000 cells from Wikipedia tables for evaluating cell filling.

Although these benchmarks have been used in various recent studies, they still suffer from a few shortcomings: (1) They are typically small sets of sampled tables with limited annotations. (2) SemTab 2019 contains a large number of instances; however, most of them are automatically generated and lack metadata/context of the Web tables. In this work, we compile a larger benchmark covering both table interpretation and table augmentation tasks. We also use some of these existing datasets for more comprehensive evaluation. By leveraging large-scale relational tables on Wikipedia and a curated KB, we ensure both the size and quality of our dataset.

4 METHODOLOGY

In this section, we introduce our TURL framework for unsupervised representation learning on relational tables. TURL is first trained on an unlabeled relational Web table corpus with pre-training objectives carefully designed to learn word semantics as well as relational knowledge between entities. The model architecture is general and can be applied to a wide range of downstream tasks with minimal modifications. Moreover, the pre-training process alleviates the need for large-scale labeled data for each downstream task.

4.1 Model Architecture

Figure 2 presents an overview of TURL which consists of three modules: (1) an embedding layer to convert different components of an input table into input embeddings, (2) N stacked structure-aware Transformer [38] encoders to capture the textual information and relational knowledge, and (3) a final projection layer for pre-training objectives. Figure 3 shows an input-output example.

4.2 Embedding Layer

Given a table $T=(C, H, E, e_t)$, we first linearize the input into a sequence of tokens and entity cells by concatenating the table metadata and scanning the table content row by row. The embedding layer then converts each token in C and H and each entity in E and e_t into an embedding representation.

Input token representation. For each token w, its vector representation is obtained as follows:

$$\mathbf{x}^{\mathrm{t}} = \mathbf{w} + \mathbf{t} + \mathbf{p}. \tag{1}$$

Here $\mathbf{w}$ is the word embedding vector, $\mathbf{t}$ is called the type embedding vector and aims to differentiate whether token w is in the table caption or a header, and $\mathbf{p}$ is the position embedding vector that provides relative position information for a token within the caption or a header.

Input entity representation. For each entity cell $e = (e^{\mathrm{e}}, e^{\mathrm{m}})$ (same for topic entity e_t), we fuse the information from the linked entity e^{e} and entity mention e^{m} together, and use an additional type embedding vector $\mathbf{t}^{\mathrm{e}}$ to differentiate three types of entity cells (i.e., subject/object/topic entities). Specifically, we calculate the input entity representation $\mathbf{x}^{\mathrm{e}}$ as:

$$\mathbf{x}^{\mathrm{e}} = \texttt{LINEAR}([\mathbf{e}^{\mathrm{e}}; \mathbf{e}^{\mathrm{m}}]) + \mathbf{t}^{\mathrm{e}}; \tag{2}$$

$$\mathbf{e}^{\mathrm{m}} = \texttt{MEAN}(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_j, \ldots). \tag{3}$$

Here $\mathbf{e}^{\mathrm{e}}$ is the entity embedding learned during pre-training. To represent entity mention e^{m}, we use its average word embedding $\mathbf{w}_j$'s. LINEAR is a linear layer to fuse $\mathbf{e}^{\mathrm{e}}$ and $\mathbf{e}^{\mathrm{m}}$.

A sequence of token and entity representations ($\mathbf{x}^{\mathrm{t}}$'s and $\mathbf{x}^{\mathrm{e}}$'s) are then fed into the next module of TURL, a structure-aware Transformer encoder, which will produce contextualized representations.

4.3 Structure-aware Transformer Encoder

We choose Transformer [38] as our base encoder block, since it has been widely used in pre-trained language models [15, 33] and achieves superior performance on various natural language processing tasks [39]. Due to space constraints, we only briefly introduce the conventional Transformer encoder and refer readers to [38] for more details. Finally, we present a detailed explanation on our proposed visibility matrix for modeling table structure.

Each Transformer block is composed of a multi-head self-attention layer followed by a point-wise, fully connected layer [38]. Specifically, we calculate the multi-head attention as follows:

$$\begin{aligned} \mathrm{MultiHead}(\mathbf{h}) &= [\mathrm{head}_1; \ldots; \mathrm{head}_{\mathrm{i}}; \ldots; \mathrm{head}_{\mathrm{k}}] W^O; \\ \mathrm{head}_{\mathrm{i}} &= \mathrm{Attention}\left(\mathbf{h}W_i^Q, \mathbf{h}W_i^K, \mathbf{h}W_i^V\right); \\ \mathrm{Attention}(Q, K, V) &= \mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d}} M\right) V. \end{aligned} \tag{4}$$

Here $\mathbf{h} \in \mathbb{R}^{n \times d_{\mathrm{model}}}$ is the hidden state output from the previous Transformer layer or the input embedding layer and n is the input sequence length. $\frac{1}{\sqrt{d}}$ is the scaling factor. $W_i^Q \in \mathbb{R}^{d_{\mathrm{model}} \times d}$,$W_i^K \in \mathbb{R}^{d_{\mathrm{model}} \times d}$,$W_i^V \in \mathbb{R}^{d_{\mathrm{model}} \times d}$ and $W^O \in \mathbb{R}^{\mathrm{k}d \times d_{\mathrm{intermediate}}}$ are parameter matrices. For each head, we have $d = d_{\mathrm{model}}/\mathrm{k}$, where k is the number of attention heads. $M \in \mathbb{R}^{n \times n}$ is the visibility matrix which we detail next.

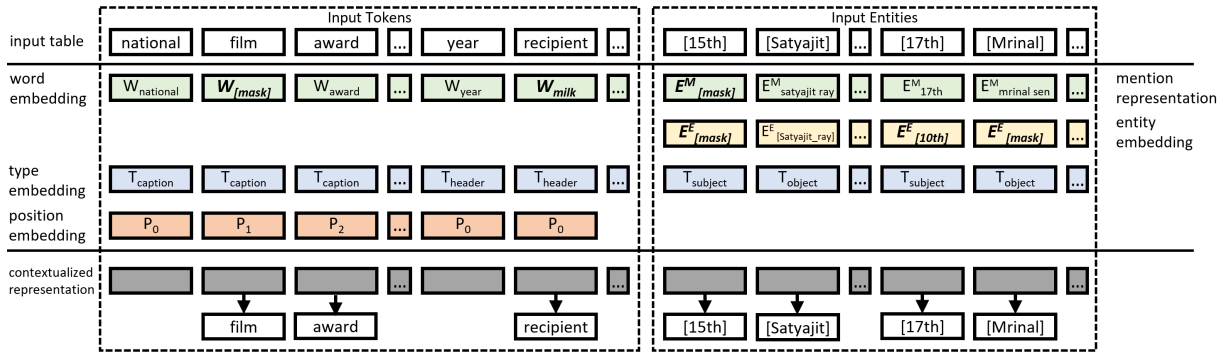


Figure 3: Illustration of the model input-output. The input table is first transformed into a sequence of tokens and entity cells, and processed for structure-aware Transformer encoder as described in Section 4.4. We then get contextualized representations for the table and use them for pre-training. Here [15th] **(which means** 15th National Film Awards**),** [Satyajit]**, ... are linked entity cells.**

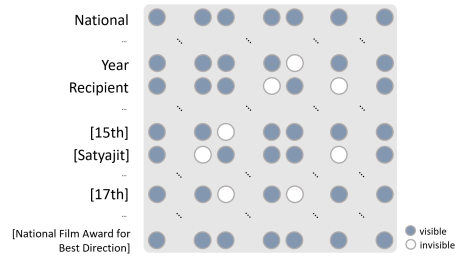


Figure 4: Graphical illustration of visibility matrix (symmetric).

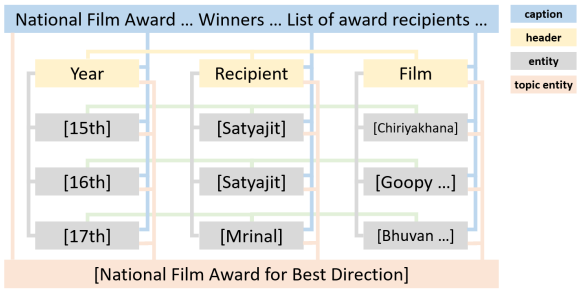


Figure 5: Graphical illustration of masked self-attention by our visibility matrix. Each token/entity in a table can only attend to its directly connected neighbors (shown as edges here).

Visibility matrix. To interpret relational tables and extract the knowledge embedded in them, it is important to model row-column structure. For example, in Figure 1, [Satyajit] and [Chiriyakhana] are related because they are in the same row, which implies that [Satyajit] directs [Chiriyakhana]. In contrast, [Satyajit] should not be related to [Pratidwandi]. Similarly, [Hindi] is a "*language*" and its representation has little to do with the header "*Film*". We propose a visibility matrix M to model such structure information in a relational table. Figure 4 shows an example of M.

Our visibility matrix acts as an attention mask so that each token (or entity) can only aggregate information from other structurally related tokens/entities during the self-attention calculation. M is a symmetric binary matrix with $M_{i,j} = 1$ if and only if $\texttt{element}_j$ is visible to $\texttt{element}_i$. The `element` here can be a token in the caption or a header, or an entity in a table cell. Specifically, we define M as follows:

- If $\texttt{element}_i$ is the topic entity or a token in table caption, $\forall j, M_{i,j} = 1$. *Table caption and topic entity are visible to all components of the table.*
- If $\texttt{element}_i$ is a token or an entity in the table and $\texttt{element}_j$ is a token or an entity in the same row or the same column, $M_{i,j} = 1$. *Entities and text content in the same row or the same column are visible to each other.*

Example 4.1. Use Figure 5 as an example. "*National Film ... recipients ...*" are tokens from the caption and [National Film Award for Best Direction] is the topic entity, and hence they can aggregate information from all other `elements`. "*Year*" is a token from a column header, and it can attend to all `elements` *except for* entity cells not belonging to that column. [Satyajit] is an entity from a table cell, so it can only attend to the caption, topic entity, entity cells in the same row/column as well as the header of that column.

4.4 Pre-training Objective

In order to pre-train our model on an unlabeled table corpus, we adopt the Masked Language Model (MLM) objective from BERT to learn representations for tokens in table metadata and propose a Masked Entity Recovery (MER) objective to learn entity cell representations.

Masked Language Model. We adopt the same Masked Language Model objective as BERT, which trains the model to capture the lexical, semantic and contextual information described by table metadata. Given an input token sequence including table caption and table headers, we simply mask some percentage of the tokens at random, and then predict these masked tokens. We adopt the same percentage settings as BERT. The pre-training data processor selects 20% of the token positions at random (note, we use a slightly larger ratio compared with 15% in [15] as we want to make the pre-training more challenging). For a selected position, (1) 80% of the time we replace it with a special [MASK] token, (2) 10% of the time we replace it with another random token, and (3) 10% of the time we keep it unchanged.

Example 4.2. Figure 3 shows an example of the above random process in MLM, where (1) "*film*", "*award*" and "*recipient*" are chosen randomly, and (2) the input word embedding of "*film*" is further chosen randomly to be replaced with the embedding of [MASK],

(3) the input word embedding of "*recipient*" to be replaced with the embedding of a random word "*milk*", and (4) the input word embedding of "*award*" to remain the same.

Given a token position selected for MLM, which has a contexutalized representation $\mathbf{h}^{\mathrm{t}}$ output by our encoder, the probability of predicting its original token $w \in \mathcal{W}$ is then calculated as:

$$P(w) = \frac{\exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{t}}) \cdot \mathbf{w}\right)}{\sum_{w_k \in \mathcal{W}} \exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{t}}) \cdot \mathbf{w}_k\right)} \tag{5}$$

Masked Entity Recovery. In addition to MLM, we propose a novel Masked Entity Recovery (MER) objective to help the model capture the factual knowledge embedded in the table content as well as the associations between table metadata and table content. Essentially, we mask a certain percentage of input entity cells and then recover the linked entity based on surrounding entity cells and table metadata. This requires the model to be able to infer the relation between entities from table metadata and encode the knowledge in entity embeddings.

In addition, our proposed masking mechanism takes advantage of entity mentions. Specifically, as shown in Eqn. 2, the input entity representation has two parts: the entity embedding $\mathbf{e}^{\mathrm{e}}$ and the entity mention representation $\mathbf{e}^{\mathrm{m}}$. For some percentage of masked entity cells, we only mask $\mathbf{e}^{\mathrm{e}}$, and as such the model receives additional entity mention information to help form predictions. This assists the model in building a connection between entity embeddings and entity mentions, and helps downstream tasks where only cell texts are available.

Specifically, we propose the following masking mechanism for MER: The pre-training data processor chooses 60% of entity cells at random. Here we adopt a higher masking ratio for MER compared with MLM, because oftentimes in downstream tasks, none or few entities are given. For one chosen entity cell, (1) 10% of the time we keep both $\mathbf{e}^{\mathrm{m}}$ and $\mathbf{e}^{\mathrm{e}}$ unchanged (2) 63% (i.e., 70% of the left 90%) of the time we mask both $\mathbf{e}^{\mathrm{m}}$ and $\mathbf{e}^{\mathrm{e}}$ (3) 27% (i.e., 30% of the left 90%) of the time we keep $\mathbf{e}^{\mathrm{m}}$ unchanged, and mask $\mathbf{e}^{\mathrm{e}}$ (among which we replace $\mathbf{e}^{\mathrm{e}}$ with embedding of a random entity to inject noise in 10% of the time). Similar to BERT, in both MLM and MER we keep a certain portion of the selected positions unchanged so that the model can generate good representations for non-masked tokens/entity cells. Trained with random tokens/entities replacing the original ones, the model is robust and utilizes contextual information to make predictions rather than simply copying the input representation.

Example 4.3. Take Figure 3 as an example. [15th], [Satyajit], [17th] and [Mrinal] are first chosen for MER. Then, (1) the input mention representation and entity embedding of [Satyajit] remain the same. (2) The input mention representation and entity embedding of [15th] are both replaced with the embedding of [MASK] (3) The input entity embedding of [Mrinal] is replaced with embedding of [MASK], while the input entity embedding of [17th] is replaced with the embedding of a random entity [10th]. In both cases, the input mention representation are unchanged.

Given an entity cell selected for MER with a contexutalized representation $\mathbf{h}^{\mathrm{e}}$ output by our encoder, the probability of predicting entity $e \in \mathcal{E}$ is then calculated as follows.

$$P(e) = \frac{\exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{e}}) \cdot \mathbf{e}^{\mathrm{e}}\right)}{\sum_{e_k \in \mathcal{E}} \exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{e}}) \cdot \mathbf{e}^{\mathrm{e}}_k\right)} \tag{6}$$

In reality, considering the entity vocabulary $\mathcal{E}$ is quite large, we only use the above equation to rank entities from a given candidate set. For efficient training, we construct the candidate set with (1) entities in the current table, (2) entities that have co-occurred with those in the current table, and (3) randomly sampled negative entities.

We use a cross-entropy loss function for both MLM and MER objectives and the final pre-training loss is given as follows:

$$loss = \sum \log\left(P(w)\right) + \sum \log\left(P(e)\right), \tag{7}$$

where the sums are over all tokens and entity cells selected in MLM and MER respectively.

Table 2: Dataset statistics (per table) in pre-training.

	split	min	mean	median	max
# row	train	1	13	8	4670
	dev	5	20	12	667
	test	5	21	12	3143
# ent. columns	train	1	2	2	20
	dev	3	4	3	15
	test	3	4	3	15
# ent.	train	3	19	9	3911
	dev	8	57	34	2132
	test	8	60	34	9215

Pre-training details. In this work, we denote the number of Transformer blocks as N, the hidden dimension of input embeddings and all Transformer block outputs as d_{model}, the hidden dimension of the fully connected layer in a Transformer block as $d_{\text{intermediate}}$, and the number of self-attention heads as k. We take advantage of a pre-trained TinyBERT [22] model, which is a knowledge distilled version of BERT with a smaller size, and set the hyperparameters as follows: N = 4, $d_{\text{model}} = 312$, $d_{\text{intermediate}} = 1200$, k = 12. We initialize our structure-aware Transformer encoder parameters, word embeddings and position embeddings with TinyBERT [22]. Entity embeddings are initialized using averaged word embeddings in entity names, and type embeddings are randomly initialized. We use the Adam [24] optimizer with a linearly decreasing learning rate. The initial learning rate is 1e-4 chosen from [1e-3, 5e-4, 1e-4, 1e-5] based on our validation set. We pre-trained the model for 80 epochs.

5 DATASET CONSTRUCTION FOR PRE-TRAINING

We construct a dataset for unsupervised representation learning based on the WikiTable corpus [4], which originally contains around 1.65M tables extracted from Wikipedia pages. The corpus contains a large amount of factual knowledge on various topics ranging from sport events (e.g., Olympics) to artistic works (e.g., TV series). The following sections introduce our data construction process as well as characteristics of the dataset.

5.1 Data Pre-processing and Partitioning

Pre-processing. The corresponding Wikipedia page of a table often provides much contextual information, such as page title and section title that can aid in the understanding of a table topic. We concatenate page title, section title and table caption to obtain a comprehensive description.

In addition, each table in the corpus contains one or more header rows and several rows of table content. For tables with more than one header row, we concatenate headers in the same column to obtain one header for each column. For each cell, we obtain hyperlinks to Wikipedia pages in it and use them to normalize different entity mentions corresponding to the same entity. We treat each Wikipedia page as an individual entity and do not use additional tools to perform entity linking with an external KB. For cells containing multiple hyperlinks, we only keep the first link. We also discard rows that have merged columns in a table.

Identify relational tables. We first locate all columns that contain at least one linked cell after pre-processing. We further filter out noisy columns with empty or illegal headers (e.g., *note*, *comment*, *reference*, digit numbers, etc.). The columns left are entity-centric and are referred to as entity columns. We then identify relational tables by finding tables that have a subject column. A simple heuristic is employed for subject column detection: the subject column must be located in the first two columns of the table and contain unique entities which we treat as subject entities. We further filter out tables containing less than three entities or more than twenty columns. With this process, we obtain 670,171 relational tables.

Data partitioning. From the above 670,171 tables, we select a high quality subset for evaluation: From tables that have (1) more than four linked entities in the subject column, (2) at least three entity columns including the subject column, and (3) more than half of the cells in entity columns are linked, we randomly select 10000 to form a held-out set. We further randomly partition this set into validation/test sets via a rough 1:1 ratio for model evaluation. All relational tables not in the evaluation set are used for pre-training. In sum, we have 570171 / 5036 / 4964 tables respectively for pre-training/validation/test sets.

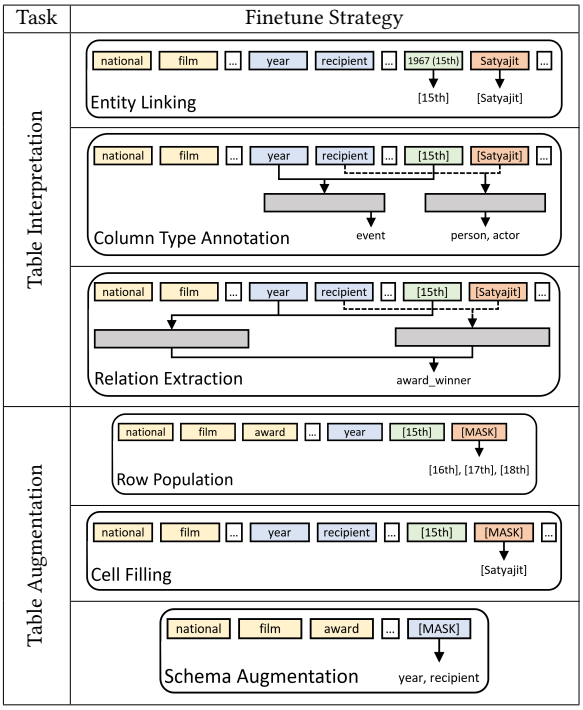
5.2 Dataset Statistics in Pre-training

Fine-grained statistics of our datasets are summarized in Table 2. We can see that most tables in our pre-training dataset have moderate size, with median of 8 rows, 2 entity columns and 9 entities per table. We build a token vocabulary using the BERT-based tokenizer [15] (with 30,522 tokens in total). For the entity vocabulary, we construct it based on the training table corpus and obtain 926,135 entities after removing those that appear only once.

6 EXPERIMENTS

To systematically evaluate our pre-trained framework as well as facilitate research, we compile a table understanding benchmark consisting of 6 widely studied tasks covering table interpretation (e.g., entity linking, column type annotation, relation extraction) and table augmentation (e.g., row population, cell filling, schema augmentation). We include existing datasets for entity linking. However, due to the lack of large-scale open-sourced datasets, we create new datasets for other tasks based on our held-out set of relational tables and an existing KB.

Table 3: An overview of our benchmark tasks and strategies to fine-tune TURL.



Next we introduce the definition, baselines, dataset and results for each task. Our pre-trained framework is general and can be fine-tuned for all the independent tasks.

6.1 General Setup across All Tasks

We use the pre-training tables to create the training set for each task, and always build data for evaluation using the held-out validation/test tables. This way we ensure that there is no overlapping tables in training and validation/test. For fine-tuning, we initialize the parameters with a pre-trained model, and further train all parameters with a task-specific objective. To demonstrate the efficiency of pre-training, we only fine-tune our model for 10 epochs unless otherwise stated.

6.2 Entity Linking

Entity linking is a fundamental task in table interpretation, which is defined as:

Definition 6.1. Given a table T and a knowledge base $\mathcal{KB}$, entity linking aims to link each potential mention in cells of T to its referent entity $e \in \mathcal{KB}$.

Entity linking is usually addressed in two steps: a candidate generation module first proposes a set of potential entities, and an entity disambiguation module then ranks and selects the entity that best matches the surface form and is most consistent with the table context. Following existing work [4, 16, 35], we focus on entity disambiguation and use an existing Wikidata Lookup service for candidate generation.

Baselines. We compare against the most recent methods for table entity linking T2K [35], Hybrid II [16], as well as the off-the-shelf Wikidata Lookup service. T2K uses an iterative matching approach that combines both schema and entity matching. Hybrid II [16] combines a lookup method with an entity embedding method. For Wikidata Lookup, we simply use the top-1 returned result as the prediction.

Fine-tuning TURL. Entity disambiguation is essentially matching a table cell with candidate entities. We treat each cell as a potential entity, and input its cell text (entity mention $\mathbf{e}^{\mathrm{m}}$ in Eqn. 2) as well as table metadata to our Transformer encoder and obtain a contextualized representation $\mathbf{h}^{\mathrm{e}}$ for each cell. To represent each candidate entity, we utilize the name and description as well as type information from a KB. The intuition is that when the candidate generation module proposes multiple entity candidates with similar names, we will utilize the description and type information to find the candidate that is most consistent with the table context. Specifically, for a KB entity e, given its name N and description D (both are a sequence of words) and types T, we get its representation $\mathbf{e}^{\mathrm{kb}}$ as follows:

$$\mathbf{e}^{\mathrm{kb}} = [\mathtt{MEAN}_{w \in N}\left(\mathbf{w}\right), \mathtt{MEAN}_{w \in D}\left(\mathbf{w}\right), \mathtt{MEAN}_{t \in T}\left(\mathbf{t}\right)]. \quad (8)$$

Here, $\mathbf{w}$ is the embedding for word w, which is shared with the embedding layer of pre-trained model. $\mathbf{t}$ is the embedding for entity type t to be learned during this fine-tuning phase. We then calculate a matching score between $\mathbf{e}^{\mathrm{kb}}$ and $\mathbf{h}^{\mathrm{e}}$ similarly as Eqn. 6. We do not use the entity embeddings pre-trained by our model here, as the goal is to link mentions to entities in a target KB, not necessarily those appear in our pre-training table corpus. The model is fine-tuned with a cross-entropy loss.

Task-specific Datasets. We use three datasets to compare different entity linking models: (1) We adopt the Wikipedia gold standards (WikiGS) dataset from [16], which contains 4,453,329 entity mentions extracted from 485,096 Wikipedia tables and links them to DBpedia [3]. (2) Since tables in WikiGS also come from Wikipedia, some of the tables may have already been seen during pre-training, despite their entity linking information is mainly used to pre-train entity embeddings (which are not used here). For a better comparison, we also create our own test set from the held-out test tables mentioned in Section 5.1, which contains 297,018 entity mentions from 4,964 tables. (3) To test our model on Web tables (i.e., those from websites other than Wikipedia), we also include the T2D dataset [25] which contains 26,124 entity mentions from 233 Web tables.[1] We use names and descriptions returned by Wikidata Lookup, and entity types from DBpedia.

The training set for fine-tuning TURL is based on our pre-training corpus, but with tables in the above WikiGS removed. We also remove duplicate entity mentions and mentions where Wikidata Lookup fails to return the ground truth entity in candidates, and finally obtain 1,264,217 entity mentions in 192,728 tables to fine-tune our model for the entity linking task.

Results. We set the maximum candidate size for Wikidata Lookup at 50 and also include the result of a Wikidata Lookup (Oracle), which considers an entity linking instance as correct if the ground-truth entity is in the candidate set. Due to lack of open-sourced implementations, we directly use the results of T2K and Hybrid II in [16]. We use F1, precision (P) and recall (R) measures for evaluation. False positive is the number of mentions where the model links to wrong entities, not including the cases where the model makes no prediction (e.g., Wikidata Lookup returns empty candidate set).

[1]We use the data released by [16] (https://doi.org/10.6084/m9.figshare.5229847).

Table 4: Model evaluation on entity linking task. All three datasets are evaluated with the same TURL + fine-tuning model.

Method	WikiGS			Our Test Set			T2D		
	F1	P	R	F1	P	R	F1	P	R
T2K [35]	34	70	22	-	-	-	82	**90**	76
Hybrid II [16]	64	69	**60**	-	-	-	**83**	85	**81**
Wikidata Lookup	57	67	49	62	62	60	80	86	75
TURL + fine-tuning	**67**	**79**	58	**68**	**71**	**66**	78	83	73
w/o entity desc.	60	70	52	60	63	58	-	-	-
w/o entity type	66	78	57	67	70	65	-	-	-
+ reweighting	-	-	-	-	-	-	82	88	77
WikiLookup (Oracle)	74	88	64	79	82	76	90	96	84

As shown in Table 4, our model gets the best F1 score and substantially improves precision on WikiGS and our own test set. The disambiguation accuracy on WikiGS is 89.62% (predict the correct entity if it is in the candidate set). A more advanced candidate generation module can help achieve better results in the future. We also conduct an ablation study on our model by removing the description or type information of a candidate entity from Eqn. 8. From Table 4, we can see that entity description is very important for disambiguation, while entity type information only results in a minor improvement. This is perhaps due to the incompleteness of DBpedia, where a lot of entities have no types assigned or have missing types.

On the T2D dataset, all models perform much better than on the two Wikipedia datasets, mainly because of its smaller size and limited types of entities. The Wikidata Lookup baseline achieved high performance, and re-ranking using our model does not further improve. However, we adopt simple reweighting[2] to take into account the original result returned by Wikidata Lookup, which brings the F1 score to 0.82. This demonstrates the potential of using features such as entity popularity (used in Wikidata Lookup) and ensembling strong base models. Additionally, we conduct an error analysis on T2D comparing our model (TURL + fine-tuning + reweighting) with Wikidata Lookup. From Table 5, we can see that while in many cases, our model can infer the correct entity type based on the context and re-rank the candidate list accordingly, it makes mistakes when there are entities in the KB that look very similar to the mentions. To summarize, Table 4 and 5 show that there is room for further improvement of our model on entity linking, which we leave as future work.

6.3 Column Type Annotation

We define the task of column type annotation as follow:

Definition 6.2. Given a table T and a set of semantic types $\mathcal{L}$, column type annotation refers to the task of annotating a column in T with $l \in \mathcal{L}$ so that all entities in the column have type l. Note that a column can have multiple types.

[2]We simply reweight the score of the top-1 prediction by our model with a factor of 0.8 and compare it with the top-1 prediction returned by Wikidata Lookup. The higher one is chosen as final prediction.

Table 5: Further analysis for entity linking on T2D corpus.

Mention	Page title	Header	Wikidata Lookup result	TURL + fine-tuning + reweighting result	Improve
philip	List of saints	Saint	Philip, male given name	Philip the Apostle, Christian saint and apostle	Yes
haycock	All 214 Wainwright fells from the pictorial guides - Wainwright Walks	Fell Name	Haycock, family name	Haycock, mountain in United Kingdom	Yes
don't you forget about me	Empty Ochestra Band - Karaoke	Name	Don't You Forget About Me, episode of Supernatural (S11 E12)	Don't You (Forget About Me), original song written and composed by Keith Forsey and Steve Schiff	Yes
bank of nova scotia	The Global 2000 - Forbes.com	Company	Scotiabank, Canadian bank based in Toronto	Bank of Nova Scotia, bank building in Calgary	No
purple finch	The Sea Ranch Association List of Birds	Common Name	Haemorhous purpureus, species of bird	Purple Finch, print in the National Gallery of Art	No

Column type annotation is a crucial task for table understanding and is a fundamental step for many downstream tasks like data integration and knowledge discovery. Earlier work [29, 35, 48] on column type annotation often coupled the task with entity linking. First entities in a column are linked to a KB and then majority voting is employed on the types of the linked entities. More recently, [10, 11, 21] have studied column type annotation based on cell texts only. Here we adopt a similar setting, i.e., use the available information in a given table directly for column type annotation without performing entity linking first.

Baselines. We compare our results with the state-of-the-art model Sherlock [21] for column type annotation. Sherlock uses 1588 features describing statistical properties, character distributions, word embeddings, and paragraph vectors of the cell values in a column. It was originally designed to predict a single type for a given column. We change its final layer to $|\mathcal{L}|$ Sigmoid activation functions, each with a binary cross-entropy loss, to fit our multi-label setting. We also evaluate our model using two datasets in [11], and include the HNN + P2Vec model as baseline. HNN + P2Vec employs a hybrid neural network to extract cell, row and column features, and combines it with property features retrieved from KB.

Fine-tuning TURL. To predict the type(s) for a column, we first extract the contextualized representation of the column $\mathbf{h}_c$ as follows:

$$\mathbf{h}_c = [\texttt{MEAN}\left(\mathbf{h}^{\text{t}}_i, \dots\right); \texttt{MEAN}\left(\mathbf{h}^{\text{e}}_j, \dots\right)]. \tag{9}$$

Here $\mathbf{h}^{\text{t}}_i$'s are representations of tokens in the column header, $\mathbf{h}^{\text{e}}_j$'s are representations of entity cells in the column. The probability of predicting type l is then given as,

$$P(l) = \text{Sigmoid}\left(\mathbf{h}_c W_l + b_l\right). \tag{10}$$

Same as with the baselines, we optimize the binary cross-entropy loss, y is the ground truth label for type l

$$loss = \sum y \log\left(P(l)\right) + (1-y)\log\left(1-P(l)\right) \tag{11}$$

Task-specific Datasets. We refer to Freebase [19] to obtain semantic types $\mathcal{L}$ because of its richness, diversity, and scale. We only keep those columns in our relational table corpus that have at least three linked entities to Freebase, and for each column, we use the common types of its entities as annotations. We further filter out types with less than 100 training instances and keep only the most representative types. In the end, we get a total number of 255 types, 628,254 columns from 397,098 tables for training, 13,025 (13,391) columns from 4,764 (4,844) tables for test (validation). We also test our model on two existing small-scale datasets, T2D-Te and Efthymiou (a subset of WikiGS annotated with types) from [11]

Table 6: Model evaluation on column type annotation task.

Method	F1	P	R
Sherlock (only entity mention) [21]	78.47	88.40	70.55
TURL + fine-tuning (only entity mention)	88.86	90.54	87.23
TURL + fine-tuning	**94.75**	**94.95**	**94.56**
w/o table metadata	93.77	94.80	92.76
w/o learned embedding	92.69	92.75	92.63
only table metadata	90.24	89.91	90.58
only learned embedding	93.33	94.72	91.97

Table 7: Accuracy on T2D-Te and Efthymiou, where scores for HNN + P2Vec are copied from [11] (trained with 70% of T2D and Efthymiou respectively and tested on the rest). We directly apply our models by type mapping without retraining.

Method	T2D-Te	Efthymiou
HNN + P2Vec (entity mention + KB) [11]	0.966	0.865
TURL + fine-tuning (only entity mention)	0.888	0.745
+ table metadata	0.860	0.904

Table 8: Accuracy on T2D-Te and Efthymiou. Here all models use T2D-Tr (70% of T2D) as training set, following the setting in [11].

Method	T2D-Te	Efthymiou
HNN + P2Vec (entity mention + KB) [11]	0.966	0.650
TURL + fine-tuning (only entity mention)	0.940	0.516
+ table metadata	0.962	0.746

and conduct two auxiliary experiments: (1) We first directly test our trained models and see how they generalize to existing datasets. We manually map 24 out of the 37 types used in [11] to our types, which results in 107 (of the original 133) columns in T2D-Te and 416 (of the original 614) columns in Efthymiou. (2) We follow the setting in [11] and use 70% of T2D as training data, which contains 250 columns.[3]

Results. For the main results on our test set, we use the validation set for early stopping in training the Sherlock model, which takes over 100 epochs. We evaluate model performance using micro F1, Precision (P) and Recall (R) measures. Results are shown in Table 6. Our model substantially outperforms the baseline, even when using the same input information (only entity mention vs Sherlock). Adding table metadata information and entity embedding learned during pre-training further boost the performance to 94.75 under F1. In addition, our model achieves such performance using only 10 epochs for fine-tuning, which demonstrates the efficiency of the pre-training/fine-tuning paradigm. More detailed results for several types are shown in Table 9, where we observe that all methods work well for coarse-grained types like `person`. However, fine-grained

[3]We use the data released by [11] (https://github.com/alan-turing-institute/SemAIDA). The number of instances is slightly different from the original paper.

Table 9: Further analysis on column type annotation: Model performance for 5 selected types. Results are F1 on validation set.

Method	person	pro_athlete	actor	location	citytown
Sherlock	96.85	74.39	29.07	91.22	55.72
TURL + fine-tuning	**99.71**	**91.14**	**74.85**	**99.32**	**79.72**
only entity mention	98.44	87.11	58.86	96.59	60.13
w/o table metadata	99.63	90.38	74.46	99.01	77.37
w/o learned embedding	99.38	90.56	71.39	98.91	75.55
only table metadata	98.26	88.80	70.86	98.11	72.54
only learned embedding	98.72	91.06	73.62	97.78	75.16

Table 10: Model evaluation on relation extraction task.

Method	F1	P	R
BERT-based	90.94	91.18	90.69
TURL + fine-tuning (only table metadata)	92.13	91.17	93.12
TURL + fine-tuning	**94.91**	**94.57**	**95.25**
w/o table metadata	93.85	93.78	93.91
w/o learned embedding	93.35	92.90	93.80

Table 11: Relation extraction results of an entity linking based system, under different agreement ratio thresholds.

Min Ag. Ratio	F1	P	R
0	68.73	60.33	**79.85**
0.4	82.10	94.65	72.50
0.5	77.68	98.33	64.20
0.7	63.10	99.37	46.23

types like `actor` and `pro_athlete` are much more difficult to predict. Specifically, it is hard for a model to predict such types for a column only based on entity mentions in cells. On the other hand, using table metadata works much better than using entity mentions (e.g., 70.86 vs 58.86 for `actor`). This indicates the importance of table context information for predicting fine-grained column types.

Results of the auxiliary experiments are summarized in Table 7 and 8. The scores shown are accuracy, i.e., the ratio of correctly labeled columns, given each column is annotated with one ground truth label. For HNN + P2Vec, the scores are directly copied from the original paper [11]. Note that in Table 7, the numbers from our models are not directly comparable with HNN + P2Vec, due to mapping the types in the original datasets to ours as mentioned earlier. However, taking HNN + P2Vec trained on in-domain data as reference, we can see that without retraining, our models still obtain high accuracy on both Web table corpus (T2D-Te) and Wikipedia table corpus (Efthymiou). We also notice that adding table metadata slightly decreases the performance on T2D while increasing that on Efthymiou, which is possibly due to the distributional differences between Wikipedia tables and general web tables. From Table 8 we can see that when trained on the same T2D-Tr split, our model with both entity mention and table metadata still outperforms or is on par with the baseline. However, when using only entity mention, our model does not perform as well as the baseline, especially when generalizing to Efthymiou. This is because: (1) Our model is pretrained with both table metadata and entity embedding. Removing both creates a big mismatch between pretraining and fine-tuning. (2) With only 250 training instances, it is easy for deep models to overfit. The better performance of models leveraging table metadata under both settings demonstrates the usefulness of context for table understanding.

6.4 Relation Extraction

Relation extraction is the task of mapping column pairs in a table to relations in a KB. A formal definition is given as follows.

Definition 6.3. Given a table T and a set of relations $\mathcal{R}$ in KB. For a subject-object column pair in T, we aim to annotate it with $r \in \mathcal{R}$ so that r holds between all entity pairs in the columns.

Most existing work [29, 35, 48] assumes that all relations between entities are known in KB and relations between columns can be easily inferred based on entity linking results. However, such methods rely on entity linking performance and suffer from KB incompleteness. Here we aim to conduct relation extraction without explicitly linking table cells to entities. *This is important as it allows the extraction of new knowledge from Web tables for tasks like knowledge base population.*

Baselines. We compare our model with a state-of-the-art text based relation extraction model [49] which utilizes a pretrained BERT model to encode the table information. For text based relation extraction, the task is to predict the relation between two entity mentions in a sentence. Here we adapt the setting by treating the concatenated table metadata as a sentence, and the headers of the two columns as entity mentions. Although our setting is different from the entity linking based relation extraction systems in [29, 35, 48], here we implement a similar system using our entity linking model described in Section 6.2, and obtain relation annotations based on majority voting of linked entity pairs, i.e., predict a relation if it holds between a minimum portion of linked entity pairs in KB (i.e., the minimum agreement ratio is larger than a threshold).

Fine-tuning TURL. We use similar model architecture as column type annotation as follows.

$$P(r) = \text{Sigmoid}\left([\mathbf{h}_c; \mathbf{h}_{c'}]W_r + b_r\right). \tag{12}$$

Here $\mathbf{h}_c, \mathbf{h}_{c'}$ are aggregated representation for the two columns obtained same as Eqn. 9. We use binary cross-entropy loss for optimization.

Task-specific Datasets. We prepare datasets for relation extraction in a similar way as the previous column type annotation task, based on our pre-training table partitions. Specifically, we obtain relations $\mathcal{R}$ from Freebase. For each table in our corpus, we pair its subject column with each of its object columns, and annotate the column pair with relations shared by more than half of the entity pairs in the columns. We only keep relations that have more than 100 training instances. Finally, we obtain a total number of 121 relations, 62,954 column pairs from 52,943 tables for training, and 2072 (2,175) column pairs from 1467 (1,560) tables for test (validation).

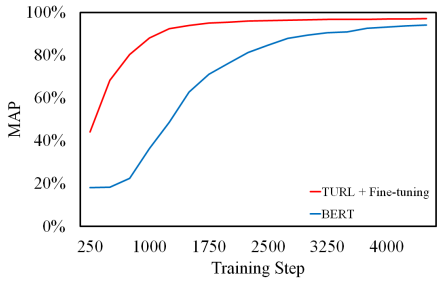


Figure 6: Comparison of fine-tuning our model and BERT for relation extraction: Our model converges much faster.

Results. We fine-tune the BERT-based model for 25 epochs. We use micro F1, Precision (P) and Recall (R) measures for evaluation. Results are summarized in Table 10.

From Table 10 we can see that: (1) Both the BERT-based baseline and our model achieve good performance, with F1 scores larger than 0.9. (2) Our model outperforms the BERT-based baseline under all settings, even when using the same information (i.e., only table metadata vs BERT-based). Moreover, we plot the mean average precision (MAP) curve on our validation set during training in Figure 6. As one can see, our model converges much faster in comparison to the BERT-based baseline, demonstrating that our model learns a better initialization through pre-training.

As mentioned earlier, we also experiment with an entity linking based system. Results are summarized in Table 11. We can see that it can achieve high precision, but suffers from low recall: The upper-bound of recall is only 79.85%, achieved at an agreement ratio of 0 (i.e., taking all relations that exist between the linked entity pairs as positive). As seen from Table 10 and 11, our model also substantially outperforms the system based on a strong entity linker.

6.5 Row Population

Row population is the task of augmenting a given table with more rows or row elements. For relational tables, existing work has tackled this problem by retrieving entities to fill the subject column [45, 47]. A formal definition of the task is given below.

Definition 6.4. Given a partial table T, and an optional set of seed subject entities, row population aims to retrieve more entities to fill the subject column.

Baselines. We adopt models from [45] and [13] as baselines. [45] uses a generative probabilistic model which ranks candidate entities considering both table metadata and entity co-occurrence statistics. [13] further improves upon [45] by utilizing entity embeddings trained on the table corpus to estimate entity similarity. We use the same candidate generation module from [45] for all methods, which formulates a search query using either the table caption or seed entities and then retrieves tables via the BM25 retrieval algorithm. Subject entities in those retrieved tables will be candidates for row population.

Fine-tuning TURL. We adopt the same candidate generation module used by baselines. We then append the [MASK] token to the input, and use the hidden representation $\mathbf{h}^e$ of [MASK] to rank these candidates as shown in Table 3. We fine-tune our model with multi-label soft margin loss as shown below:

$$P(e) = \text{Sigmoid}\left(\texttt{LINEAR}(\mathbf{h}^e) \cdot \mathbf{e}^e\right),$$
$$loss = \sum_{e \in \mathcal{E}_C} y \log\left(P(e)\right) + (1-y)\log\left(1-P(e)\right). \quad (13)$$

Here $\mathcal{E}_C$ is the candidate entity set, and y is the ground truth label of whether e is a subject entity of the table.

Table 12: Model evaluation on row population task. Recall is the same for all methods because they share the same candidate generation module.

# seed	0		1	
Method	MAP	Recall	MAP	Recall
EntiTables [45]	17.90	63.30	42.31	78.13
Table2Vec [13]	-	63.30	20.86	78.13
TURL + fine-tuning	**40.92**	63.30	**48.31**	78.13

Table 13: Model evaluation on cell filling task.

Method	P @ 1	P @ 3	P @ 5	P @ 10
Exact	51.36	70.10	76.80	84.93
H2H	51.90	70.95	77.33	85.44
H2V	52.23	70.82	77.35	85.58
TURL	**54.80**	**76.58**	**83.66**	**90.98**

Table 14: Model evaluation on schema augmentation task.

Method	#seed column labels	
	0	1
kNN	80.16	**82.01**
TURL + fine-tuning	**81.94**	77.55

Task-specific Datasets. Tables in our pre-training set with more than 3 subject entities are used for fine-tuning TURL and developing baseline models, while tables in our held-out set with more than 5 subject entities are used for evaluation. In total, we obtain 432,660 tables for fine-tuning with 10 subject entities on average, and 4,132 (4,205) tables for test (validation) with 16 (15) subject entities on average.

Results. The experiments are conducted under two settings: without any seed entity and with one seed entity. For experiments without the seed entity, we only use table caption for candidate generation. For entity ranking in EntiTables [45], we use the combination of caption and label likelihood when there is no seed entity, and only use entity similarity when seed entities are available. This strategy works best on our validation set. As shown in Table 12, our method outperforms all baselines. In particular, previous methods rely on entity similarity and are not applicable or have poor results when there is no seed entity available. Our method achieves a decent performance even without any seed entity, which demonstrates the effectiveness of TURL for generating contextualized representations based on both table metadata and content.

6.6 Cell Filling

We examine the utility of our model in filling other table cells, assuming the subject column is given. This is similar to the setting in [42, 46], which we formally define as follows.

Definition 6.5. Given a partial table T with the subject column filled and an object column header, cell filling aims to predict the object entity for each subject entity.

Table 15: Case study on schema augmentation. Here we show average precision (AP) for each example. Support Caption is the caption of the source table that kNN found to be most similar to the query table. Our model performs worse when there exist source tables that are very similar to the query table (e.g., comparing support caption vs query caption).

Method	Query Caption	Seed	Target	AP	Predicted	Support Caption
kNN	2010 santos fc season out	pos.	name, moving to	1.0	moving to, name, player, moving from, to	2007 santos fc season out
Ours				0.58	moving to, fee/notes, destination club, fee, loaned to	-
kNN	first ladies and gentlemen of panama list	no.	name, president	0.20	country, runner-up, champion, player, team team	first ladies of chile list of first ladies
Ours				0.14	year, runner-up, spouse, name, father	-
kNN	list of radio stations in metro manila am stations	name	format, covered location	1.0	format, covered location, company, call sign, owner	list of radio stations in metro manila fm stations
Ours				0.83	format, owner, covered location, city of license, call sign	-

Baselines. We adopt [46] as our base model. It has two main components, candidate value finding and value ranking. The same candidate value finding module is used for all methods: Given a subject entity e and object header h for the to-be-filled cells, we find all entities that appear in the same row with e in our pre-training table corpus , and only keep entities whose source header h' is related to h. Here we use the formula from [46] to measure the relevance of two headers $P(h'|h)$,

$$P(h'|h) = \frac{n(h', h)}{\sum_{h''} n(h'', h)}. \tag{14}$$

Here $n(h', h)$ is the number of table pairs in the table corpus that contain the same entity for a given subject entity in columns h' and h. The intuition is that if two tables contain the same object entity for a given subject entity e in columns with headings h_a and h_b, then h_a and h_b might refer to the same attribute. For value ranking, the key is to match the given header h with the source header h', we can then get the probability of the candidate entity e belongs to the cell $P(e|h)$ as follows:

$$P(e|h) = \text{MAX}\left(\text{sim}(h', h)\right). \tag{15}$$

Here h''s are the source headers associated with the candidate entity in the pre-training table corpus. $\text{sim}(h', h)$ is the similarity between h' and h. We develop three baseline methods for $\text{sim}(h', h)$: (1) **Exact**: predict the entity with exact matched header, (2) **H2H**: use the $P(h'|h)$ described above. (3) **H2V**: similar to [13], we train header embeddings with Word2Vec on the table corpus. We then measure the similarity between headers using cosine similarity.

Fine-tuning TURL. Since cell filling is very similar to the MER pre-training task, we do not fine-tune the model, and directly use [MASK] to select from candidate entities same as MER (Eqn. 6).

Task-specific Datasets. To evaluate different methods on this task, we use the held-out test tables in our pre-training phase and extract from them those subject-object column pairs that have at least three valid entity pairs. Finally we obtain 9,075 column pairs for evaluation.

Results. For candidate value finding, using all entities appearing in the same row with a given subject entity e achieves a recall of 62.51% with 165 candidates on average. After filtering with $P(h'|h) > 0$, the recall drops slightly to 61.45% and the average number of candidates reduces to 86. For value ranking, we only consider those test instances with the target object entity in the candidate set and evaluate them under Precision@K (or, P@K). Results are summarized in Table 13, from which we show: (1) Simple **Exact** match achieves decent performance, and using **H2H** or **H2V** only sightly improves the results. (2) Even though our model directly ranks the candidate entities without explicitly using their source table information, it outperforms other methods. This indicates that our model already encodes the factual knowledge in tables into entity embeddings through pre-training.

6.7 Schema Augmentation

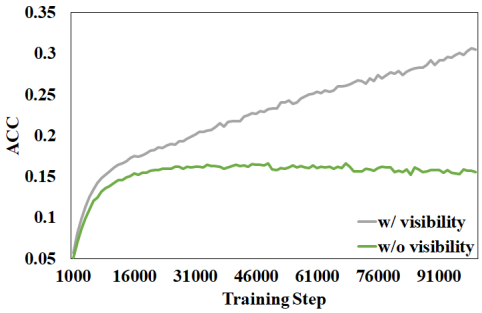
Aside from completing the table content, another direction of table augmentation focuses on augmenting the table schema, i.e., discovering new column headers to extend a table with more columns [6, 13, 42, 45]. Following [13, 42, 45], we formally define the task below.

Definition 6.6. Given a partial table T, which has a caption and zero or a few seed headers, and a header vocabulary $\mathcal{H}$, schema augmentation aims to recommend a ranked list of headers $h \in \mathcal{H}$ to add to T.

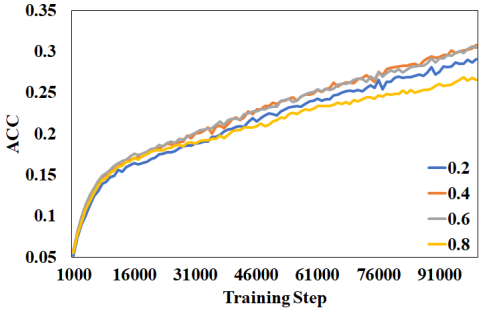
Baselines. We adopt the method in [45] which searches our pre-training table corpus for related tables, and use headers in those related tables for augmentation. More specifically, we encode the given table caption as a tf-idf vector and then use the K-nearest neighbors algorithm (kNN) [2] with cosine similarity to find the top-10 most related tables. We rank headers from those tables by aggregating the cosine similarities for tables they belong to. When seed headers are available, we re-weight the tables by the overlap of their schemas with seed headers same as [45].

Fine-tuning TURL. We concatenate the table caption, seed headers and a [MASK] token as input to our model. The output for [MASK] is then used to predict the headers in a given header vocabulary $\mathcal{H}$. We fine-tune our model use binary cross-entropy loss.

Task-specific Datasets. We collect $\mathcal{H}$ from the pre-training table corpus. We normalize the headers using simple rules, only keep those that appear in at least 10 different tables, and finally obtain 5652 unique headers, with 316,858 training tables and 4,646 (4,708) test (validation) tables.



(a) Effect of visibility matrix.



(b) Effect of different MER mask ratios.

Figure 7: Ablation study results.

Results. We fine-tune our model for 50 epochs for this task, based on the performance on the validation set. We use mean average precision (MAP) for evaluation.

From Table 14, we observe that both kNN baseline and our model achieve good performance. Our model works better when no seed header is available, but does not perform as well when there is one seed header. We then conduct a further analysis in Table 15 using a few examples: One major reason why kNN works well is that there exist tables in the pre-training table corpus that are very similar to the query table and have almost the same table schema. On the other hand, our model oftentimes suggests plausible, semantically related headers, but misses the ground-truth headers.

6.8 Ablation Study

In this section, we examine the effects of two important designs in TURL: the visibility matrix and MER with different mask ratios. During the pre-training phase, at each training step, we evaluate TURL on the validation set for *object entity prediction*. We choose this task because it is similar to the cell filling downstream task and it is convenient to conduct during pre-training (e.g., ground-truth is readily available and no need to modify the model architecture, etc.).

Given a table in our validation set, we predict each object entity by first masking the entity cell (both $\mathbf{e}^{\mathrm{e}}$ and $\mathbf{e}^{\mathrm{m}}$) and obtaining a contextualized representation of the [MASK] (which attends to the table caption, corresponding header, as well as other entities in the same row/column before the current cell position) and then applying Eqn. 6. We compare the top-1 predicted entity with the ground truth and show the accuracy (ACC) on average. Results are summarized in Figure 7.

Figure 7a clearly demonstrates the advantage of our visibility matrix design. Without the visibility matrix (an element can attend to every other element during pre-training), it is hard for the model to capture the most relevant information (e.g., relations between entities) in the table for prediction. From Figure 7b, we observe that at a mask ratio of 0.8, the objective entity prediction performance drops in comparison with other lower ratios. This is because this task requires the model to not only understand the table metadata, but also learn the relation between entities. A high mask ratio forces the model to put more emphasis on the table metadata, while a lower mask ratio encourages the model to leverage the relation between entities. Meanwhile, a very low mask ratio such as 0.2 also hurts the pre-training performance, because only a small portion of entity cells are actually used for training in each iteration. A low mask ratio also creates a mismatch between pre-training and fine-tuning, since for many downstream tasks, only few seed entities are given. Considering both aspects as well as that the results are not sensitive w.r.t. this parameter, we set the MER mask ratio at 0.6 in pre-training.

7 CONCLUSION

This paper presents a novel pre-training/fine-tuning framework (TURL) for relational table understanding. It consists of a structure-aware Transformer encoder to model the row-column structure as well as a new Masked Entity Recovery objective to capture the semantics and knowledge in relational Web tables during pre-training. On our compiled benchmark, we show that TURL can be applied to a wide range of tasks with minimal fine-tuning and achieves superior performance in most scenarios. Interesting future work includes: (1) Focusing on other types of knowledge such as numerical attributes in relational Web tables, in addition to entity relations. (2) Incorporating the rich information contained in an external KB into pre-training.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments. Authors at the Ohio State University were sponsored in part by Google Faculty Award, the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, NSF CAREER #1942980, Fujitsu gift grant, and Ohio Supercomputer Center [9]. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

REFERENCES

[1] Ahmad Ahmadov, Maik Thiele, Julian Eberius, Wolfgang Lehner, and Robert Wrembel. 2015. Towards a hybrid imputation approach using web tables. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*. IEEE, 21–30.

[2] Naomi S. Altman. 1992. An Introduction to Kernel and Nearest Neighbor Nonparametric Regression.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *ISWC/ASWC*.

[4] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *Proceedings of the 14th International Conference*

on The Semantic Web-ISWC 2015-Volume 9366. 425–441.

[5] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*.

[6] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment* 1, 1 (2008), 538–549.

[7] Michael J. Cafarella, Alon Y. Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. 2008. Uncovering the Relational Web. In *11th International Workshop on the Web and Databases, WebDB 2008, Vancouver, BC, Canada, June 13, 2008*.

[8] Matteo Cannaviccio, Lorenzo Ariemma, Denilson Barbosa, and Paolo Merialdo. 2018. Leveraging wikipedia table schemas for knowledge graph augmentation. In *Proceedings of the 21st International Workshop on the Web and Databases*. 1–6.

[9] Ohio Supercomputer Center. 1987. Ohio Supercomputer Center. http://osc.edu/ark:/19495/f5s1ph73

[10] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles A. Sutton. 2018. ColNet: Embedding the Semantics of Web Tables for Column Type Prediction. In *AAAI*.

[11] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles A. Sutton. 2019. Learning Semantic Annotations for Tabular Data. In *IJCAI*.

[12] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding Related Tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 817–828.

[13] Lei Min Deng, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *SIGIR'19*.

[14] Xiang Deng and Huan Sun. 2019. Leveraging 2-hop Distant Supervision from Table Entity Pairs for Relation Extraction. *arXiv preprint arXiv:1909.06007* (2019).

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

[16] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. 2017. Matching Web Tables with Knowledge Base Entities: From Entity Lookups to Entity Embeddings. In *International Semantic Web Conference*.

[17] Raul Castro Fernandez and Samuel Madden. 2019. Termite: a system for tunneling through heterogeneous data. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–8.

[18] Xavier Glorot, Antoine Bordes, Jason Weston, and Yoshua Bengio. 2013. A semantic matching energy function for learning with multi-relational data. *Machine Learning* 94 (2013), 233–259.

[19] Google. 2015. Freebase Data Dumps. https://developers.google.com/freebase/data.

[20] Jonathan Herzig, P. Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TAPAS: Weakly Supervised Table Parsing via Pre-training. In *ACL*.

[21] Madelon Hulsebos, Kevin Zeng Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, cCaugatay Demiralp, and C'esar A. Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019).

[22] Xiaoqi Jiao, Y. Yin, Lifeng Shang, Xin Jiang, Xusong Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling BERT for Natural Language Understanding. *ArXiv* abs/1909.10351 (2019).

[23] Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, and Kavitha Srinivas. 2020. SemTab 2019: Resources to Benchmark Tabular Data to Knowledge Graph Matching Systems. In *European Semantic Web Conference*. Springer, 514–530.

[24] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

[25] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A Large Public Corpus of Web Tables containing Time and Context Metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*. 75–76.

[26] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *arXiv preprint arXiv:2004.00584* (2020).

[27] LimayeGirija, SarawagiSunita, and ChakrabartiSoumen. 2010. Annotating and searching web tables using entities, types and relationships. In *VLDB 2010*.

[28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[29] Varish Mulwad, Timothy W. Finin, Zareen Syed, and Anupam Joshi. 2010. Using Linked Data to Interpret Tables. In *COLD*.

[30] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*.

[31] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL-HLT*.

[32] Matthew E Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. 2019. Knowledge Enhanced Contextual Word Representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 43–54.

[33] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[34] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. 2013. Relation Extraction with Matrix Factorization and Universal Schemas. In *HLT-NAACL*.

[35] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching HTML Tables to DBpedia. In *WIMS '15*.

[36] Yoones A Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. [n.d.]. Knowledge Base Augmentation using Tabular Data.

[37] Qi shan Wang, Zhendong Mao, Biwu Wang, and Li Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering* 29 (2017), 2724–2743.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[39] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 353–355.

[40] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zhigang Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*.

[41] Jason Weston, Antoine Bordes, Oksana Yakhnenko, and Nicolas Usunier. 2013. Connecting Language and Knowledge Bases with Embedding Models for Relation Extraction. *ArXiv* abs/1307.7973 (2013).

[42] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 97–108.

[43] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *NeurIPS*.

[44] Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*.

[45] Shuo Zhang and Krisztian Balog. 2017. Entitables: Smart assistance for entity-focused tables. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 255–264.

[46] Shuo Zhang and Krisztian Balog. 2019. Auto-completion for data cells in relational tables. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 761–770.

[47] Shuo Zhang and Krisztian Balog. 2020. Web Table Extraction, Retrieval, and Augmentation: A Survey. *ACM Transactions on Intelligent Systems and Technology (TIST)* 11 (2020), 1 – 35.

[48] Ziqi Zhang. 2017. Effective and efficient Semantic Table Interpretation using TableMiner+. *Semantic Web* 8 (2017), 921–957.

[49] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced Language Representation with Informative Entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 1441–1451.