AFRL-RI-RS-TR-2024-010

QUIC-M - A SYSTEM FOR QUALITY-AWARE INTERACTIVE CURATION OF MODELS

BROWN UNIVERSITY

JANUARY 2024

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2024-010 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

GENNADY R. STASKEVICH
Work Unit Manager

/ S /

JULIE BRICHACEK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
JANUARY 2024	FINAL TECHNICAL REPORT	JULY 2022	JULY 2023

4. TITLE AND SUBTITLE

QUIC-M - A SYSTEM FOR QUALITY-AWARE INTERACTIVE CURATION OF MODELS

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
N/A	FA8750-17-2-0102	62702E
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
N/A	N/A	R27Q

6. AUTHOR(S)

Eliezer Upfal

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Brown University Box 1910 Providence RI 02912	

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Research Laboratory/RISC 525 Brooks Road Rome NY 13441-4505	AFRL/RI	AFRL-RI-RS-TR-2024-010

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The goal of a DARPA Data Driven Discovery of Models (D3M) program is to develop tools and processes that will enable subject matter experts to apply automated data science methods for the creation and curation of machine learning models for extracting latent information from real, complex data.

15. SUBJECT TERMS

Interactive Data Exploration, Human-in-the-loop Data Exploration, Automatic Machine Learning, Interactive Data Exploration, Visualization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	20

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
GENNADY STASKEVICH	N/A

Table of Contents

List of Figures

1.0 Summary

In this project, we built new breed of systems designed to enable interactive data analytics and machine learning (ML). The main goal was to develop the first system for Quality-aware Interactive Curation of Models, as part of TA2 and TA3. We enabled domain experts to build models themselves without the need to involve a data scientist. First, we extended our automatic selection of algorithms from a model family and hyper-parameter tuning techniques implemented in ML to fully-automatically build and test a large set of models. Second, we integrated these techniques into our interactive human-in-the-loop data exploration and model building suite that built on a novel pen-and-touch interface that allows domain experts to curate ML workflows in an intuitive, fluid way. Third, we developed novel risk evaluation techniques, which continuously monitor users and warn them about potential wrong conclusions or models. The foundational research was carried at Brown University and MIT. The accumulation of this work has resulted in the formation of a start-up company, Einblick Inc., and the development of a product.

2.0 Introduction

Democratizing Data Science requires a fundamental rethinking of the way data analytics and model discovery is done. Available tools for analyzing massive data sets and curating machine learning models (e.g., R and Spark) are limited in a number of fundamental ways. First, existing tools require well-trained data scientists to select the appropriate techniques to build models and to evaluate their outcomes. Second, existing tools require heavy data preparation steps (e.g., building indexes, cleaning data form errors) and are often too slow to give interactive feedback to domain experts in the model building process, severely limiting the possible interactions. Third, current tools do not provide adequate analysis of statistical risk factors in the model development.

In this project, we built is a novel multi-modal notebook that allows users to build data workflows using Python, SQL, and no-code, making it a powerful tool for technical, yet approachable for less technical users. It addresses many common shortcomings of traditional notebooks, such as reproducibility of results, reactivity of cells, sharing of cells and results, as well as real-time collaboration. It allows users to seamlessly switch to a canvas view, which supports branching, side-by-side comparison, similar to modern collaboration tools (e.g., Figma, Miro, Mural, etc.). Combining TA2 and TA3 work under the Data-Driven Discovery of Models (D3M) program, we first extended our automatic selection of algorithms from a model family and hyper-parameter tuning techniques implemented in ML to fully-automatically build and test a large set of models. Second, we integrated these techniques into our interactive human-in-the-loop data exploration and model building suite that built on a novel pen-and-touch interface that allows domain experts to curate ML workflows in an intuitive, fluid way. Third, we developed novel risk evaluation techniques, which continuously monitor users and warn them about potential wrong conclusions or models.

The final deliverable of this project is a new start-up company, Einblick, which is a Brown University/MIT spin-off and based on 8 years of research and three entire rewrites of the platform to build a tailored experience for interactive data science and analytics. Our goal has always been to make data science and analysis more enjoyable, more collaborative experience. Building on the work in this project, Einblick product provides a next-generation, AI-native, multi-modal data notebook to build workflows and data apps. We believe that generative AI will fundamentally change the way people do analytics and we

explore how natural language models can be integrated into Einblick and deeply embedded large language models within our novel interface.

Figure 1: Einblick workspace in canvas mode, showing non-linearity, branching, code, no-code & input control cells, and natural language integration.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Design Principles

In the pursuit of creating the next-generation computational notebook experience, the following seven design principles guide our approach:

- Customer experience comes first: Building a high-quality product requires putting the customer experience, specifically the UX, at the front and center, and building infrastructure to support it, rather than the other way around.
- UI excellence for computational notebooks: Design and engineer with precision, and combine UI elegance with the computational power of notebook to build highly intuitive and user-friendly interface, catering to data scientists/analysts and their stakeholders.
- Collaboration and sharing: Treat sharing and support for real-time collaboration as a first principle.
- Multi-Modality: Offer native support for multiple programming languages commonly used in data science and analysis, such as Python and SQL, provide no/low-code options to accelerate common workflows and let users mix and match these modalities.
- Embrace AI: Leverage AI to boost users' efficiency and assist them in getting work done.
- Extensibility: Design with extensibility in mind, allowing users to easily integrate and share custom functionality.

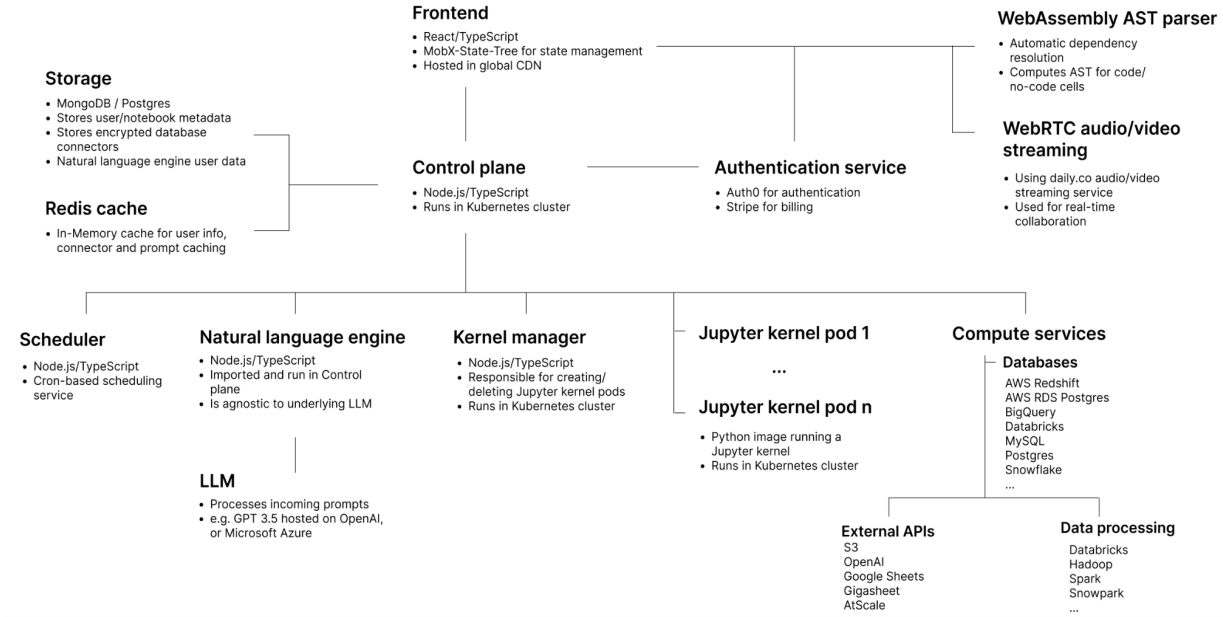
- Security and Privacy: Diligently follow best practices for handling our users' data and securing workspaces against potential vulnerabilities.

By strictly adhering to these design principles, we were able to achieve a better user experience, which wouldn't have been possible by "just" extending Python notebooks.

3.2 Technology

3.2.1 System Overview

Einblick is a cloud-based application consisting of various components and services that power the unique user experience in the frontend.



1

Figure 2: System design overview of Einblick[1]

The core functionality is powered by the *frontend*, *control plane*, *kernel manager* and *Jupiter kernel pods* components The frontend is a multi-package TypeScript/React bundle hosted in a global CDN, while control plane, kernel manager and Jupyter kernels are scalable services deployed to a Kubernetes cluster using Helm-Chart. User code (Python/SQL code and no-code cells) run in dedicated Jupyter kernels, one per canvas shared with all users in that canvas. SQL queries are executed using SQLAlchemy from within the kernel. The kernel manager is responsible for creating new and deleting idle kernels via Kubernetes API (@kubernetes/client-node package). The frontend communicates with the Kernel via the control plane using WebSockets and a REST API. This has several benefits: 1) The control plane has access to the user's connector credentials, while the frontend does not, for security reasons. When running a code to fetch data from a database, for instance, the frontend sends a code template that's then filled in by the

[1] This document does not go into details about each individual component/service but we're happy to elaborate as part of the discussion.

control plane. 2) The kernels are not publicly exposed, and any access to them is authenticated by the control plane and its authentication service. 3) The control plane enforces access controls for workspaces such as view-only or organization-wide edit access. These checks are applied when accessing the kernel. 4) the control plane can directly store and broadcast the response of an execution (which can be big in size), rather than offloading this task to a potentially slow client, and 5) all executions can be logged in a central place.

3.2.2 Layouting Engine

When users interact with the system, they operate on *workspaces*. At rest, a workspace is a self-contained JSON (BSON) document that describes its content, such as, code and no-code cells added by the user, runtime configurations, package requirements, bookmarks, secrets, etc.


```
{
  "_id": "64bd5eb3ea4bda1d8580b242",
  "CreatedAt": 1690132147814,
  "UpdatedAt": 1690403838209,,
  "OwnerUserId": "auth0|6780ae31-ab34-4ae6-91f4-fb9976075e63",
  "GlobalShareType": "OrganizationView",
  "SharedWith": [],
  "Name": "Untitled (204)",
  "Cells": {},
  "Computations": {},
  "Zones": {},
  "RootComputations": {},
  "Bookmarks": {},
  "InitCellScript": "",
  "Secrets": {},
  "Runtime": {
    "Id": "bd34189f-a7fca6ae",
    "KernelId": "997bba65-05ea2188",
    "RequestedKernelPythonVersion": "3.10"
  },
  "NotebookLayout": {
    "CellIndices": {},
  },
  "GridOptions": {
    "ShowGridlines": false,
    "SnapToGrid": false,
    "GridSize": 50,
    "GutterRatio": 0
  },
  "CanFork": false,
  "ShowAllDependencies": true,
  "IgnoreImportDependencies": true
}
```


Figure 3: Abbreviated example of the structure of a workspace document.

When a user loads a workspace, the frontend fetches the workspace JSON via the control plane from the server and parses and converts it to its corresponding MobX state tree representation. The layout engine then renders its content based on the user's setting. It supports two view modes: *linear,* a linear notebook view that's equivalent to traditional computation notebooks like Jupyter, and *canvas*, a 2D

environment where users can freely lay out their analyses, arrange cells side-by-side, and use pan/zoom to navigate the space and present results.
Users can seamlessly switch between the two modes.

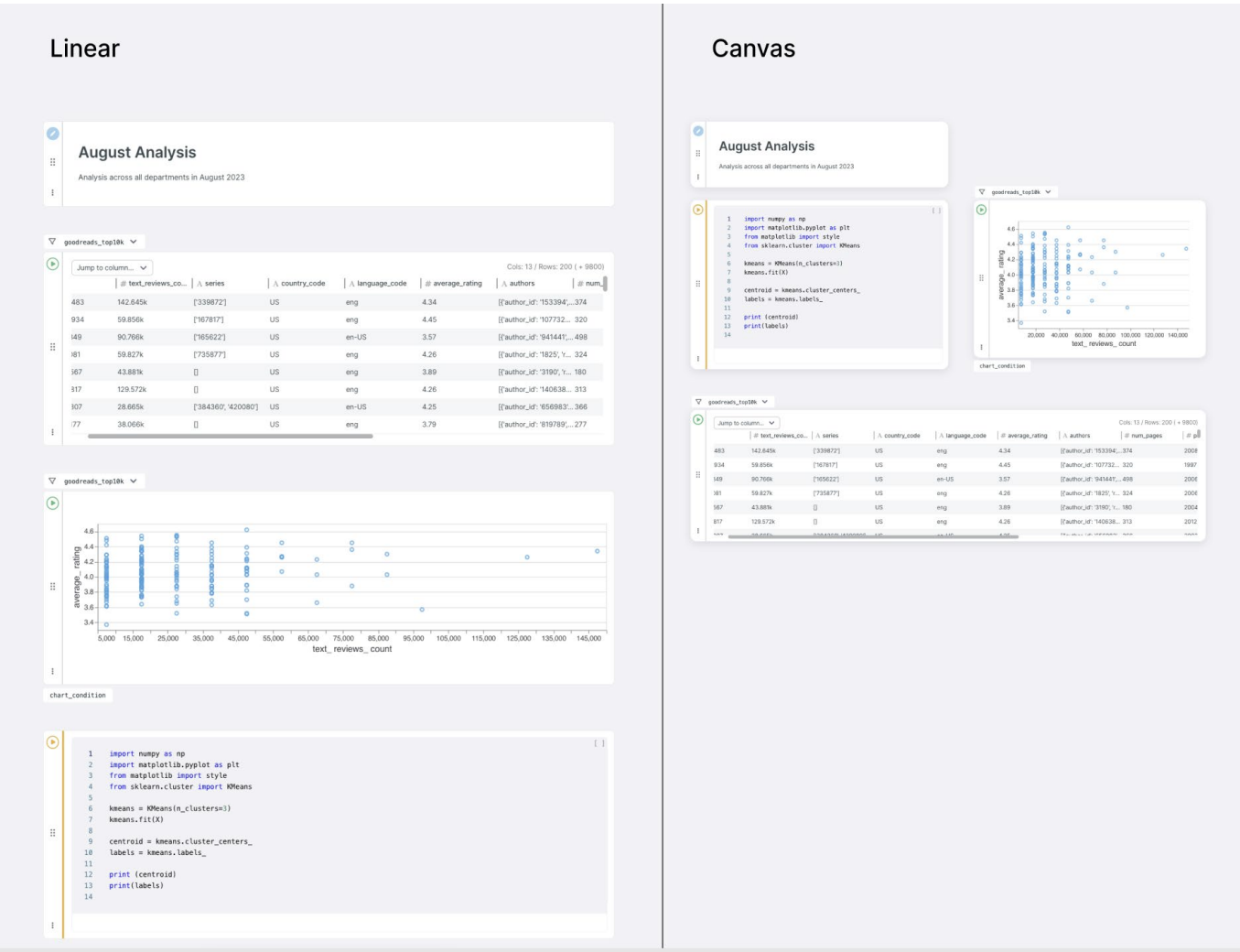


Figure 4: EInblick supports two different layout modes: linear and canvas

3.2.3 Multi-modality

We support four different executable types of cells: *Python*, *SQL, Markdown*, and *no-code*. Python, Markdown, and SQL cells feature an Monaco-based code-editor with advanced editing functionality, code completion, documentation hints, keyboard short-cuts, etc., that can easily be integrated with LLM-based code-completion services.

Python cells are Jupyter compatible and support all common output types, such as plain text, images, HTML and markdown, as well as STDIN input through a user control. Python cells don't have any restrictions and give the user access to the full file system as well as to installing/managing packages using pip, a popular Python package installer.

SQL cells are built on top of Python cells and allow users to 1) run SQL directly on in-memory dataframes using DuckDB, and 2) select a data connector and run SQL on an external database. The SQL cell features a built-in schema viewer where users can inspect the tables and their contents of a database. The schema viewer can easily be integrated with data cataloging and/or data governance systems.

Markdown cells can be used to render markdown to HTML, allowing users to include formatted text, images and videos.

No-code cells can be thought of as thin UI wrappers around Python cells which expose the parameters of a script in a designated input panel (see screenshot below). *No-code cells are not hard-coded,* but rather declaratively specified as a combination of a JSON manifest and code-template, and an optional renderer. The manifest specifies properties such as the name of the no-code cell type, visual rendering properties, the number of required input dataframes, as well as all parameters required for the code to run. Parameters can be derived from data, e.g., a parameter can be a column name of the input dataframe, or they can be data-agnostic constants, such as a numeric value or string. In either case, no-code cells will parse these specs and automatically generate and render an appropriate UI. When a user runs a no-code cell, the system combines the user-specified parameters with the code template to assemble an executable Python script.
No-code cells can have custom renderers, such as the chart, table, filter cell. The renderers are React components that are provided access to the data and other utility functions to parse and render the results returned by the kernel.

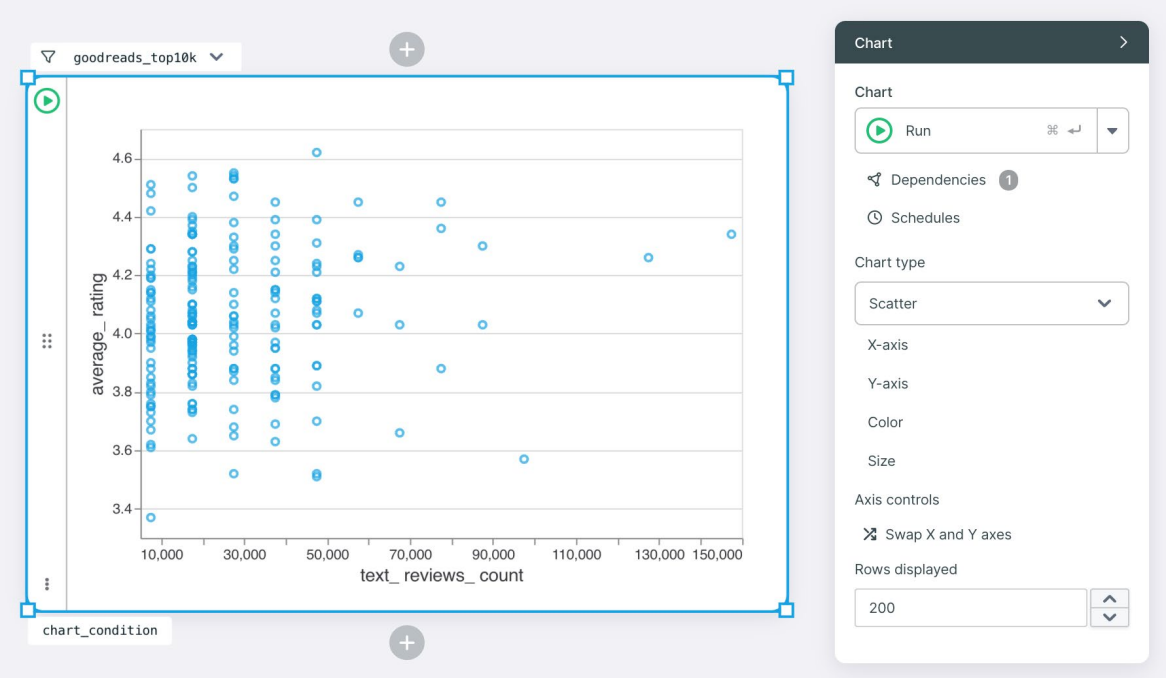


Figure 5: A no-code Chart cell that can be configured through the properties panel on the right. The UI is generated based on the cell manifest.

3.2.4 Cell dependency analysis and reactivity

In traditional notebooks, cells are either executed individually, top-down by running all cells, or all preceding cells of a selection. One of the most disliked aspects of traditional notebooks is the reliance on these semantics because it makes it difficult to understand the dependencies between cells, i.e. which cells need to be re-run in order to reproduce a result, and to figure out which cells need to be re-run when a parameter changes.

Our solution to this problem is that we automatically compute cell dependency graphs by analyzing the contents of code and no-code cells and display the results to the user. As shown in the screenshot below, every cell visually declares its output and input dependencies and we visualize the relationship between cells with links. Additionally, we can leverage this dependency graph to visually indicate which cells need to be rerun if a parameter changes.

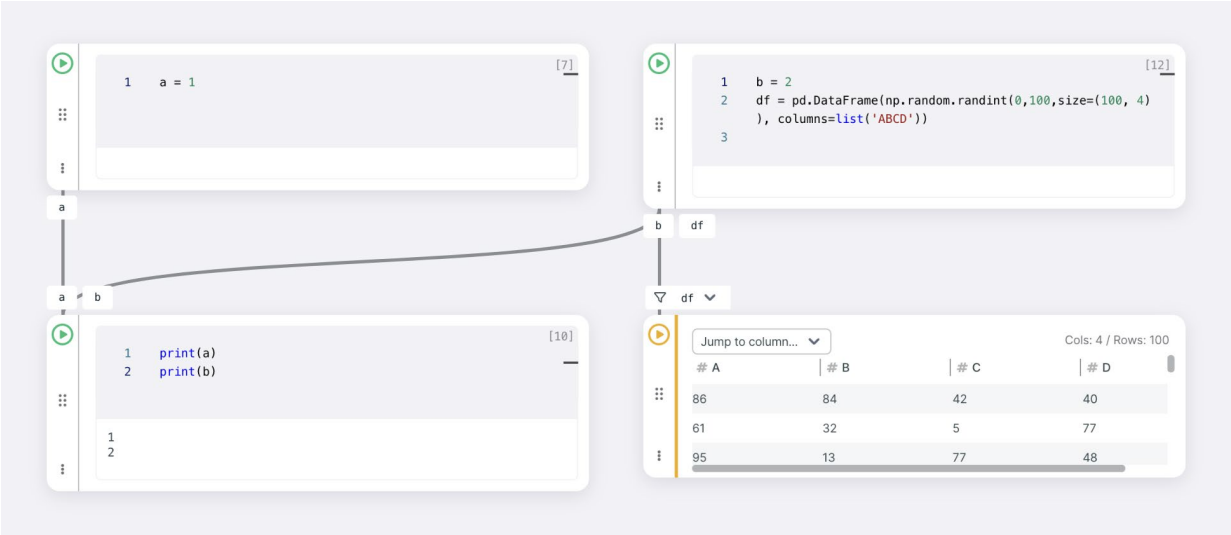


Figure 6: Illustration of how Einblick automatically detects input and output dependencies, and marks “stale” cells (orange).

We identify the variables defined and referenced within each cell and resolve ambiguities and cyclic dependencies by leveraging properties of the linear/canvas layout, considering factors such as proximity or user interaction history (which cells were last interacted with). Users can always override and enforce specific cell dependencies if necessary. This allows us to model each flow of cells as a directed acyclic graph, eliminating the reliance on execution order and ensuring full reproducibility of all paths.

The dependency analysis module is implemented by running static code analysis on all code and no-code cells. The key to making this feature work well is speed because dependencies need to be updated on every keystroke or parameter interaction. We use a proprietary algorithm that leverages an extremely fast WebAssembly package to compute an *abstract syntax tree* for every cell. Using its output our algorithm derives variables and imports based on a set of rules which are then translated into a dependency graph for each cell.

Since we know which inputs any cell relies on, we can further leverage this concept to expose a *reactive cell* execution mode, which is particularly valuable for interactive data apps. It allows users to create

interactive applications without defining and worrying about complicated event handling or callbacks, can be used for caching mechanisms to optimize app performance, and guarantees that script recomputation is triggered only when necessary.

Finally, as we explain in detail in our natural language engine document, being able to use the cell dependencies is crucial for computing relevant context for our LLM integration.

3.2.5 Cell execution, scheduling

As outlined in the previous section, in traditional notebooks, cells are executed *individually*, *top-down* by running all cells, or all cells *above a selection*. Einblick builds on top of these execution semantics and lets users run all dependencies of a cell, before running the cell itself. We call this “run flow”.

The advantage of “run flow” is that to reproduce a result of a cell, only relevant predecessors are run. Our algorithm intelligently linearizes a dependency graph into a sequence of cell executions that are then sent to the kernel (a nice side-effect of this linearization is that any flow can be exported to a plain Python script or .ipynb file). When a flow is executed in the kernel, our system aggregates the results for each cell and returns them to the frontend.

Additionally, users can schedule the execution of cells through a convenient UI accessible through the properties panel of a cell. Internally, running a cell on a schedule uses “run flow”, to make sure that all its predecessors, including calls to fetch data from a data source, for instance, are re-executed.

3.2.6 Reusable UI components

In our UI we use a mix of styled off-the-shelf primitive (blueprintjs) and a large number of battle-tested custom components that we’ve built over the years. A few of these components include

- Data table: A data table that supports common operations like sorting, grouping, aggregating. Rows and columns in the data table are fully virtualized and scale up to millions of rows
- Chart: A flexible charting component that uses Vega-lite under the hood
- Filter: A filter component that supports arbitrarily nested Boolean expression in an easy-to-use UI
- Schema viewer: A database schema explorer that can be used to inspect tables and views, and copy relevant pieces the SQL editor
- Code editor: A Monaco-based code-editor with advanced editing functionality, code completion, documentation hints, keyboard short-cuts, etc., that can easily be integrated with LLM-based code-completion services liker
- And many more

3.2.7 Authentication, Sharing and Permissions

We built an authentication service that integrates with third-party authentication providers. For our SASS deployment we use Auth0 , for on-prem deployment we use Keycloak, both of which integrate with identity providers like Google, GitHub, LDAP, Active Directory etc. Users are authenticated using JWT tokens.

Once authenticated, users can create and access a number of first class entities of Einblick, such as Workspace, Data Connector, and a Schedule. Each entity is owned by a user, and can be assigned organization-wide or fine-grained access permissions. Options include

- Restricted view or edit access
- Anyone with the link can view
- Anyone with the link can edit
- A mix of anyone with the link can view/edit and restricted access

Authentication and access control is handled in the control plane.

3.2.8 State Management

Einblick's frontend is complex and needs to keep track of a lot of state, which poses a significant challenge for scalability and rendering performance. On a high-level the UI state is divided into three state trees: persistent application state, volatile application state, and volatile UI state.

The persistent application state is a MobX state tree that comprises the entire data model, from users, workspaces, data connectors, to apps and plugins, etc.. For efficiency, the state tree is loaded from multiple collections in MongoDB, lazily, on demand. The system embraces MobX's reactive programming model: the persistent state tree is fully observable, meaning any modification of a state that's represented in the frontend automatically re-renders the minimal set of components to represent the state change. Similarly, we leverage the ability to detect fine-grained changes on the data model to emit RFC 6902 JSON-patches that represent the state change and send it to the control plane for persistence, and store it locally in memory for undo/redo.

```
{
        "op": "replace",
        "path: "/address",
        "value": "Stark Tower LA"
}
```

Figure 7: Example of RFC 6902 JSON patch that the frontend produces when the persistent application state changes. We send these patches to the control plane for persistence and broadcasting to other users.

The domain/data model for the persistent state is fully tailored to the needs of Einblick's extensible UI adding new cell types or plugins does not require any changes to the data model.

The volatile state tree represents state which does not need to be persisted but is required for the business logic in the application. Finally, the UI state tree is represented by the React component tree which itself holds the state of controlled components used for conditional rendering.

3.2.9 Real-time Collaboration

Einblick offers real time collaboration, enabling users to see fellow participants and their cursors and track their actions, such as interacting with cells, moving, resizing, writing code, changing parameters of no-code cells etc. Users can also switch to “observe” mode to follow the participant's cursor.

Real-time collaboration is implemented using fine-grained state changes (outlined in a previous section) from our state management layer, which are sent to the control plane via WebSocket. The control plane manages a connection pool by workspace: changes are broadcast to all authenticated users in the same workspace. We made sure that real-time collaboration scales smoothly. Since multiple control plane instances could serve the same workspace, all changes are broadcast by publishing the event in a dedicated Redis channel.

State changes are automatically classified as transient or persistent. Transient updates such as intermediate cursor updates are not persisted, but broadcast to all connected clients, whereas persistent updates are both broadcast and persisted. We are currently using a last-writer-wins conflict resolution protocol, which we may replace with a Conflict-free Replicated Data Type (CRDT) protocol in the future.

3.2.10 Rendering performance

Key to our unique UX is smooth rendering performance. This poses a particularly big challenge because of a number of hard constraints. Using HTML/JS/CSS can perform poorly on large DOMs. But, using plain WebGL is not a viable option because it would significantly impact compatibility with Jupyter notebooks, e.g., the textinput-heavy nature of notebooks would require us to build alternatives to standard input controls such as Monaco-based code-editors, and to the heterogeneous, interactive outputs which cells need to be able render.
The solution we’ve come up with through years of experimentation scales smoothly to workspaces containing many hundreds of cells. It uses a mix of leveraging a set of JS and CSS instructions to decide which DOM nodes to store on as texture GPU, diligently avoiding any re-rendering while textures are being transformed, and ensuring that re-renders outside of a textures subtree are kept to a minimum while interactions are being performed.

3.2.11 Kernel Lifecycle

Kernels are managed by Kernel Manager, a light-weight component that interfaces with both the control plane and Kubernetes via TRPC/@kubernetes/client-node respectively to instantiate, monitor, restart, and shutdown iPython kernels on demand. The kernel manager spins up Jupyter images by default with the desired Python version and provision them with resources based on the user's subscription. While right now users cannot use their own images, this would be easy to support in the future. The kernel manager monitors idle kernels and shuts them down if needed. Currently we don’t persist kernels but due to reproducible flows it’s easy to re-instantiate its state. For the future we’re considering serializing the kernels state and encrypting it at rest for paying customers.

3.2.12 Persistence

All application state is persisted in a MongoDB instance that runs in our kubernetes cluster. We store data for all top level entities such as users, workspaces, data connectors, apps, and plugins, etc. in separate collections, and update them by using a custom component that translates JSON patches to Mongo queries. All sensitive fields like data connector details are stored as encrypted binary fields. While

we're currently using MongoDB, using a different key/value store or switching to a relational DB would be straight-forward.

Aside from our main MongoDB document store, we maintain a set of databases for product analytics (A snowflake DB that gets synced through Segment and an AWS RDS Postgres database for telemetry of our natural language engine.

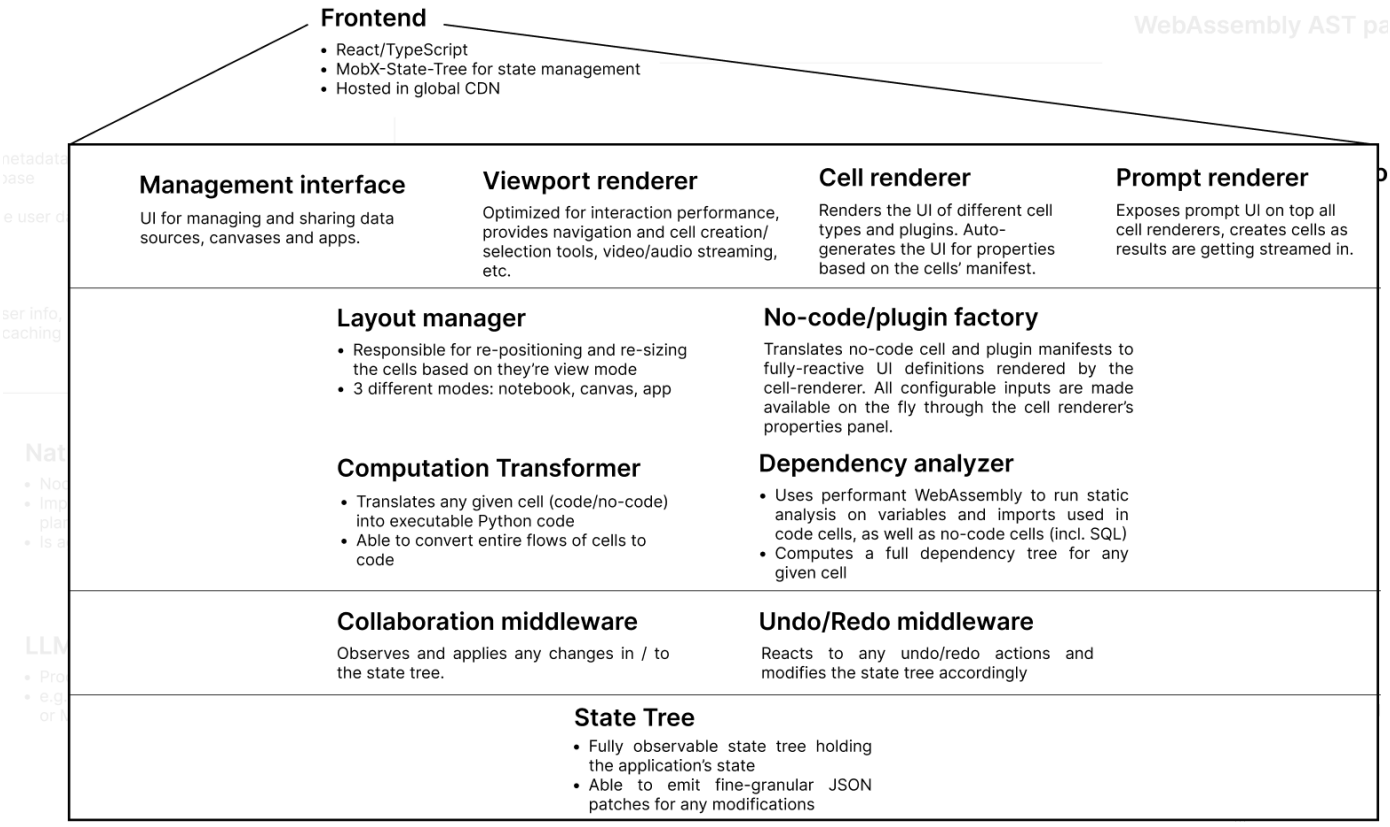


Figure 8: Architecture overview front end

4.0 RESULTS AND DISCUSSION

The main deliverables of this project were publications, patent application, and software.

The ultimate goal of this project was to "Democratizing Data Science" by providing an interactive curation of ML pipelines easily accessible to domain expertise with only basic ML skills. Standard tools for analyzing massive data sets and curating machine learning models are limited in a number of fundamental ways. First, existing tools require well-trained data scientists to select the appropriate techniques to build models and to evaluate their outcomes. Second, existing tools require heavy data preparation steps and are often too slow to give interactive feedback to domain experts in the model

building process, severely limiting the possible interactions. Third, current tools do not provide adequate analysis of statistical risk factors in the model development. The goal was to enable domain experts to build the machine learning pipelines an order of magnitude faster than machine learning experts while having model qualities comparable to expert solutions.

Our first major deliverable was the QuIC-M design for an interactive human- in-the-loop data exploration and model building suite [6]. QuIC-M used best practice rules to generate the search space of pipelines, and employed cost models to select promising pipelines. Further, it included an adaptive pipeline selection algorithm to traverse this search space.

The second deliverable of this project was Alpine *Meadow* [9], a fully Interactive Automated Machine Learning tool. What made this system unique is not only the focus on interactivity, but also the combined systemic and algorithmic design approach; on one hand we leveraged ideas from query optimization, on the other we devised novel selection and pruning strategies combining cost-based Multi-Armed Bandits and Bayesian Optimization.

AutoML has been widely used by domain experts without machine learning knowledge to extract actionable insights out of data. However, previous studies only emphasize the high accuracy of the final answer, which can take several hours, if not a couple of days to complete. What makes our system unique is not only the focus on interactivity, but also the combined systemic and algorithmic design approach. We design novel algorithms for AutoML search and co-design the execution runtime to efficiently execute the ML workloads. We evaluate our system on over 300 datasets and compare against other AutoML tools, as well as expert solutions. Not only is Alpine Meadow able to significantly outperform the other AutoML systems while — in contrast to the other systems — providing interactive latencies, but also outperforms in 80% of the cases expert solutions over data sets we have never seen before. On April 2019, Alpine Meadow was ranked first in DARPA performed D3M Automatic Machine Learning competition.

The third major deliverable of this project is Davos [19], Einblick' s new design and packaging of the system to provide a multi-modal notebook. The system allows users to build data workflows using Python, SQL, and no-code, making it a powerful tool for technical, yet approachable for less technical users. It addresses many common shortcomings of traditional notebooks, such as reproducibility of results, reactivity of cells, sharing of cells and results, as well as real-time collaboration. While Davos can be used in linear fashion, like a traditional notebook, it allows users to seamlessly switch to a canvas view, which supports branching, side-by-side comparison, similar to modern collaboration tools.

Another deliverables of this projects are Niseko [12], an open-source large scale datasets for meta-learning dataset with more than 5 million pipelines and 50 million pipeline runs on 300 datasets with clear-defined structures and easy-to-use APIs; and IDE Bench [10], a new benchmark called IDEBench, designed to evaluate database engines based on common IDE workflows and metrics that matter to the end-user.

Finally, in a sequence of works we addressed the fundamental question of accuracy of data driven discoveries, in particular when combined with multifaceted visualization tools. We formulate the problem in terms of statistical multi-comparison problems, study rigorous solutions, and implemented and tested them in our system [1,2,3,4,7].

5.0 CONCLUSIONS

Recently, a new horizon in data analytics, prescriptive analytics, is becoming more and more important to make data-driven decisions. As opposed to the progress of democratizing data acquisition and access, making data-driven decisions remains a significant challenge for people without technical expertise. In this regard, existing tools for data analytics which were designed decades ago still present a high bar for domain experts, and removing this bar requires a fundamental rethinking of both interface and backend.

In this project we developed a new approach to Interactive Automated Machine Learning, with low-latency automatic selection, based on Multi-Armed Bandits, Bayesian Optimization and Meta-Learning theory, to efficiently explore the pipeline search space and enables domain experts to bring value to the optimization process. We have tested our system on datasets with very heterogeneous characteristics, from sample size to feature types. Our experiments show that when compared against state-of-the-art systems or expert solutions, our novel system generally generates better results in a shorter amount of time.

6.0 REFERENCES

APPENDIX A - Publications and Presentations

1. Carsten Binnig, Lorenzo De Stefani, Tim Kraska, Eli Upfal, Emanuel Zgraggen, Zheguang Zhao: Toward Sustainable Insights, or Why Polygamy is Bad for You. CIDR 2017; **Keywords: Risk-awareness, Multi-Hypothesis Testing, Interactive Data Exploration**
2. Yue Guo, Carsten Binnig, Tim Kraska: What you see is not what you get!: Detecting Simpson's Paradoxes during Data Exploration. HILDA@SIGMOD 2017; **Keywords: Risk-awareness, Interactive Data Exploration**
3. Zheguang Zhao, Lorenzo De Stefani, Emanuel Zgraggen, Carsten Binnig, Eli Upfal, Tim Kraska: Controlling False Discoveries During Interactive Data Exploration. SIGMOD Conference 2017; **Keywords: Risk-awareness, Multi-Hypothesis Testing, Interactive Data Exploration**
4. Zheguang Zhao, Emanuel Zgraggen, Lorenzo De Stefani, Carsten Binnig, Eli Upfal, Tim Kraska: Safe Visual Data Exploration. SIGMOD Conference 2017; **Keywords: Risk-awareness, Multi-Hypothesis Testing, Interactive Data Exploration**
5. Carsten Binnig, Joseph M. Hellerstein, Aditya G. Parameswaran: Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2017, Chicago, IL, USA, May 14, 2017 .ACM 2017, ISBN 978-1-4503-5029-7; **Keywords: Human-in-the-loop Data Exploration**
6. Carsten Binnig, Benedetto Buratti, Yeounoh Chung, Cyrus Cousins, Dylan Ebert, Tim Kraska, Zeyuan Shang, Isabella Tromba, Eli Upfal, Linnan Wang, Robert Zeleznik, Emanuel Zgraggen: Towards Interactive Curation & Automatic Tuning of ML Pipelines, 1st Inaugural Conference on Systems ML 2018; **Keywords: Automatic and Interactive Machine Learning**
7. Emanuel Zgraggen, Zheguang Zhao, Robert Zeleznik, Tim Kraska: Investigating the Effect of the Multiple Comparisons Problem in Visual Analysis, CHI 2018; **Keywords: Risk, Interactive Data Exploration, Statistics, Visualization**
8. Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H. Chi, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, Vikram Nathan: SageDB: A Learned Database System. CIDR 2019. **Keywords: Data Ingestion and Curation, Interactive Data Analysis, Visualization**
9. Zeyuan Shang, Emmanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Yeounoh Chung, Phillipp Eichmann, Carsten Binnig, Eli Upfal, Tim Kraska: Democratizing Data Science through Interactive Curation of ML Pipelines. SIGMOD 2019. **Keywords: Automated Machine Learning, Visualization**
10. Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, Tim Kraska: IDEBench: A new Benchmark for Interactive Data Exploration. SIGMOD 2020. **Keywords: Benchmarking for Interactive Data Exploration**
11. Tobias Ziegler, Matthias Jasny, Tim Kraska, Carsten Binnig: DB4ML: An In-Memory Database Kernel with Machine Learning Support. SIGMOD 2020. **Keywords: Machine Learning in a DBMS**
12. Zeyuan Shang, Emanuel Zgraggen, Philipp Eichmann, Tim Kraska: Niseko: a Large-Scale Meta-Learning Dataset, Workshop on Meta-Learning at NeurIPS 2019 **Keywords: Meta Learning**

13. Zeyuan Shang, Emanuel Zgraggen, Tim Kraska: Alpine Meadow: A System for Interactive AutoML, Workshop on Systems for ML at NeurIPS 2019 **Keywords: Automatic and Interactive Machine Learning**
14. Carsten Binnig, Benedetto Buratti, Yeounoh Chung, Cyrus Cousins, Tim Kraska, Zeyuan Shang, Eli Upfal, Robert C. Zeleznik, Emanuel Zgraggen: Towards Interactive Curation & Automatic Tuning of ML Pipelines. DEEM@SIGMOD 2018; **Keywords: Automatic and Interactive Machine Learning**
15. Moritz Kulessa, Alejandro Molina, Carsten Binnig, Benjamin Hilprecht, Kristian Kersting: Model-based Approximate Query Processing. AIDB@VLDB. **Keywords: Interactive Data Exploration and Data Cleaning**
16. Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, Steven Euijong Whang: Slice Finder: Automated Data Slicing for Model Validation. ICDE 2019: 1550-1553. **Data and Model Quality**
17. Tim Kraska: Northstar: An Interactive Data Science System. PVLDB 11(12): 2150-2164 (2018) **Keywords: Automatic and Interactive Machine Learning**
18. Lorenzo De Stefani, Leonhard F. Spiegelberg, Eli Upfal, and Tim Kraska: VizCertify: A framework for secure visual data exploration. DSAA 2019. **Keywords: Visual Data Exploration**
19. Shang, Zeyuan; Zgraggen, Emanuel; Buratti, Benedetto; Eichmann, Philipp; Karimeddiny, Navid; Meyer, Charlie; Runnels, Wesley; Kraska, Tim: Davos: a system for interactive data-driven decision making" published at VLDB 14 (12): 2893-2905 (2021). **Keywords: Automatic and Interactive Machine Learning**

7. LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

SQLAlchemy	https://www.sqlalchemy.org
JSON (BSON)	https://www.mongodb.com/json-and-bson
Jupyter kernel	https://docs.jupyter.org/en/latest/index.html
DuckDB	https://duckdb.org
WebAssembly	https://webassembly.org
blueprintjs	https://blueprintjs.com
Auth0	https://auth0.com