

REPORT DOCUMENTATION PAGE			Form Approved OMB NO. 0704-0188		
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 07-07-2017		2. REPORT TYPE Final Report		3. DATES COVERED (From - To) 1-Jul-2013 - 31-Mar-2017	
4. TITLE AND SUBTITLE Final Report: Fundamental Theory and Parallel Inference for Probabilistic Programming (10.3.1 Integrated Intelligence)			5a. CONTRACT NUMBER W911NF-13-1-0212		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER 611102		
6. AUTHORS Joshua Tenenbaum, Vikash Mansinghka			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAMES AND ADDRESSES Massachusetts Institute of Technology (MIT) 77 Massachusetts Avenue NE18-901 Cambridge, MA 02139 -4307			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) 63850-NS.31		
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited					
13. SUPPLEMENTARY NOTES The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.					
14. ABSTRACT This project was conceived with three major goals: (i) develop new probabilistic programming technology that addressed limitations of first-generation languages such as Church; (ii) demonstrate the capabilities of knowledge-based AI systems written using these new probabilistic programming languages, emphasizing reflective uses of probabilistic programming; and (iii) develop mathematical theory that addresses fundamental questions associated with probabilistic programs. Over the past four years, we have accomplished all three of these goals.					
15. SUBJECT TERMS probabilistic programming, artificial intelligence, data science, autonomous systems					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Joshua Tenenbaum
a. REPORT UU	b. ABSTRACT UU	c. THIS PAGE UU			19b. TELEPHONE NUMBER 617-452-2010

RPPR
as of 13-Sep-2017

Agency Code:

Proposal Number:

Agreement Number:

Organization:

Address: , ,

Country:

DUNS Number:

EIN:

Date Received:

Report Date:

for Period Beginning and Ending

Title:

Begin Performance Period:

End Performance Period:

Report Term: -

Submitted By:

Email:

Phone:

Distribution Statement: -

STEM Degrees:

STEM Participants:

Major Goals:

Accomplishments:

Training Opportunities:

Results Dissemination:

Plans Next Period:

Honors and Awards:

Protocol Activity Status:

Technology Transfer:

Executive Summary	2
Goal 1 - Probabilistic Programming Systems	3
BayesDB	5
Picture	6
Venture	6
Goal 2: AI Capabilities	8
Capability #1: Probabilistic programs for inferring the probable goals of autonomous agents	8
Capability #2: Probabilistic programs for inferring 3D scene structure from single images	12
Capability #3 - AI-assisted data science for time series	18
Capability #4: AI-assisted data science for databases	22
Goal 3 - Fundamental Theory	26
Bibliography	31

Executive Summary

This project was conceived with three major goals: (i) develop new probabilistic programming technology that addressed limitations of first-generation languages such as Church; (ii) demonstrate the capabilities of knowledge-based AI systems written using these new probabilistic programming languages, emphasizing reflective uses of probabilistic programming; and (iii) develop mathematical theory that addresses fundamental questions associated with probabilistic programs. Over the past four years, we have accomplished all three of these goals.

Our research has yielded open-source probabilistic programming systems that have led to commercial deployments, sponsorship from other defense and intelligence sources, and multiple long-term academic collaborations. Our models of visual scene understanding and intuitive physics have had a significant impact on multiple AI fields, including winning an award at the leading computer vision conference (CVPR 2015), and inspiring follow-on projects at major industry AI labs (Microsoft Research, DeepMind, Facebook AI, Google).

We have been invited to give a high-profile tutorial on this research at the upcoming NIPS conference, which is expected to have an audience of thousands of participants from across industry, academia and government, as well as a keynote at the O'Reilly AI conference and executive briefings at Intel and Microsoft. At the same time, this research opens the door to even more valuable opportunities for advancing both the basic science of human cognition and the engineering of robust autonomous systems.

The main body of this report gives technical data and results on four key accomplishments chosen based on relevance for ARO. The first three are AI capability demonstrations that reflect advances towards goals #1 and #2, while goal #3 reflects advances in fundamental theory.

1. Probabilistic programs for inferring the probable goals of people, drones, and cars from observations of their motion. (*Goals 1 and 2*)
2. Probabilistic programs for inferring 3D scene structure from single images. (*Goals 1 and 2*)
3. AI-assisted data science via probabilistic programs that synthesize and query other probabilistic programs. (*Goals 1 and 2*)
4. New theory and algorithms for estimating the accuracy of a broad class of approximate inference algorithms. (*Goal 3*)

Goal 1 - Probabilistic Programming Systems

Probabilistic modeling and inference have become central tools in modern computing. They provide a rigorous mathematical framework for interpreting data in light of modeling assumptions that leave room for uncertainty and ambiguity. Probabilistic modeling and inference are used in both real-time applications, such as inferring the probable location of self-driving cars from noisy sensor data and incomplete maps, and offline computations, such as identifying gene boundaries from sequence data. The models themselves are often produced by applying probabilistic inference algorithms to identify probable models from large spaces of possibilities, given sufficient data.

The emerging field of probabilistic programming aims to formalize and automate aspects of probabilistic modeling and inference by integrating key ideas from probability theory with programming languages. Some probabilistic programming languages have thousands of users already. For example, Stan is a domain-specific probabilistic language for hierarchical Bayesian modeling, and is regularly used for data analysis in ecology and astrophysics. Other probabilistic languages have been used to produce prototypes based on modeling and inference that may not have been feasible to develop manually. For example, the BLOG language has been used to build a sensor fusion system for monitoring compliance with the Comprehensive Nuclear Test-Ban Treaty, interpreting data from hundreds of seismic, radionuclide, and hydroacoustic sensors from around the globe. A recent four-year program funded by the US Defense Advanced Research Projects Agency (DARPA) has catalyzed the development of several other languages and evaluated some of the productivity gains they can deliver. For example, some of these languages have been used to solve hard problems from computer vision (Kulkarni, T.D., Kohli, P., Tenenbaum, J.B., and Mansinghka, V., 2015), computer graphics (Richie, 2014), and data science (Saad, Casarsa, and Mansinghka, 2017; Saad and Mansinghka, 2016; Schaechtle, Saad, Radul, and Mansinghka, 2017) in under 100 lines of code.

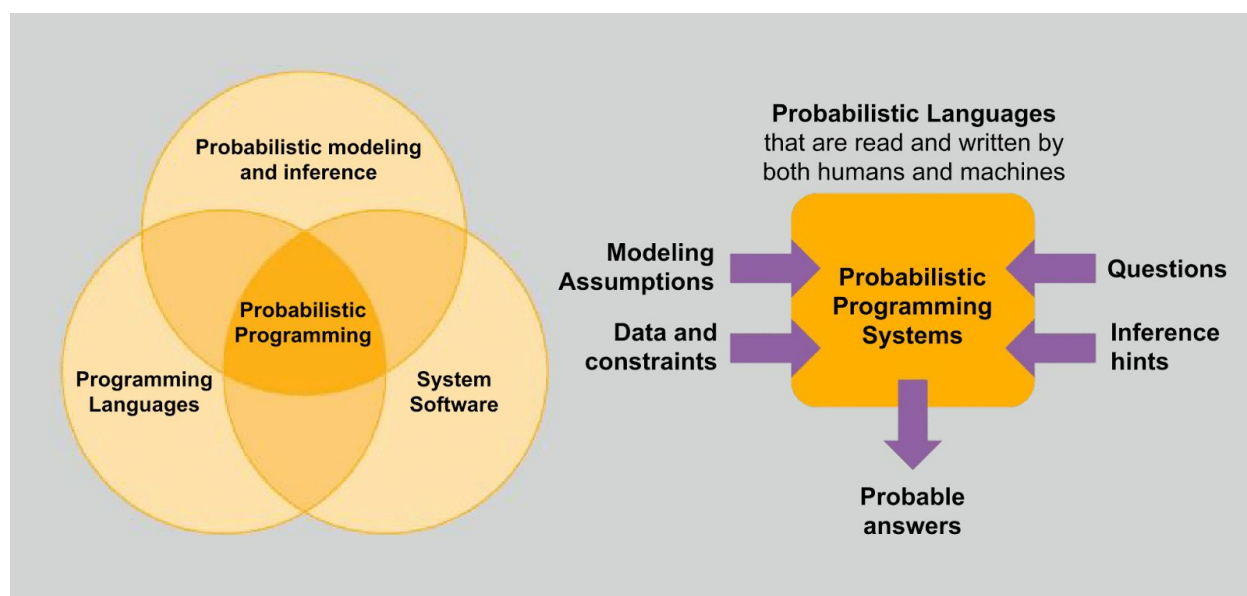


Figure 1. An overview of probabilistic programming. Probabilistic programming is an emerging field at the intersection of probabilistic modeling and inference, programming languages, and systems software.

1. **Represent *models as programs***
 programs that generate samples & programs that calculate densities
2. **Represent *inference and learning as meta-programs***
 programs that take programs as inputs and/or produce programs as outputs
3. **Some MIT languages (BayesDB, Gen.jl) support *programmable inference***

Figure 2. Key technical ideas in probabilistic programming. Probabilistic programming languages enable users to (i) represent models using programs and (ii) represent key operations on models using meta-programs. One distinctive capability of the probabilistic programming languages we developed is that they enable users to customize the meta-programs used for learning and inference.

BayesDB

BayesDB is a probabilistic programming platform that aims to enable users to query the probable implications of their data as directly as SQL databases enable them to query the data itself. By combining ordinary SQL with three new primitives — SIMULATE, INFER, and ESTIMATE — users of BayesDB can detect predictive relationships between variables, retrieve statistically similar data items, identify anomalous data points and variables, infer missing values, and synthesize hypothetical subpopulations.

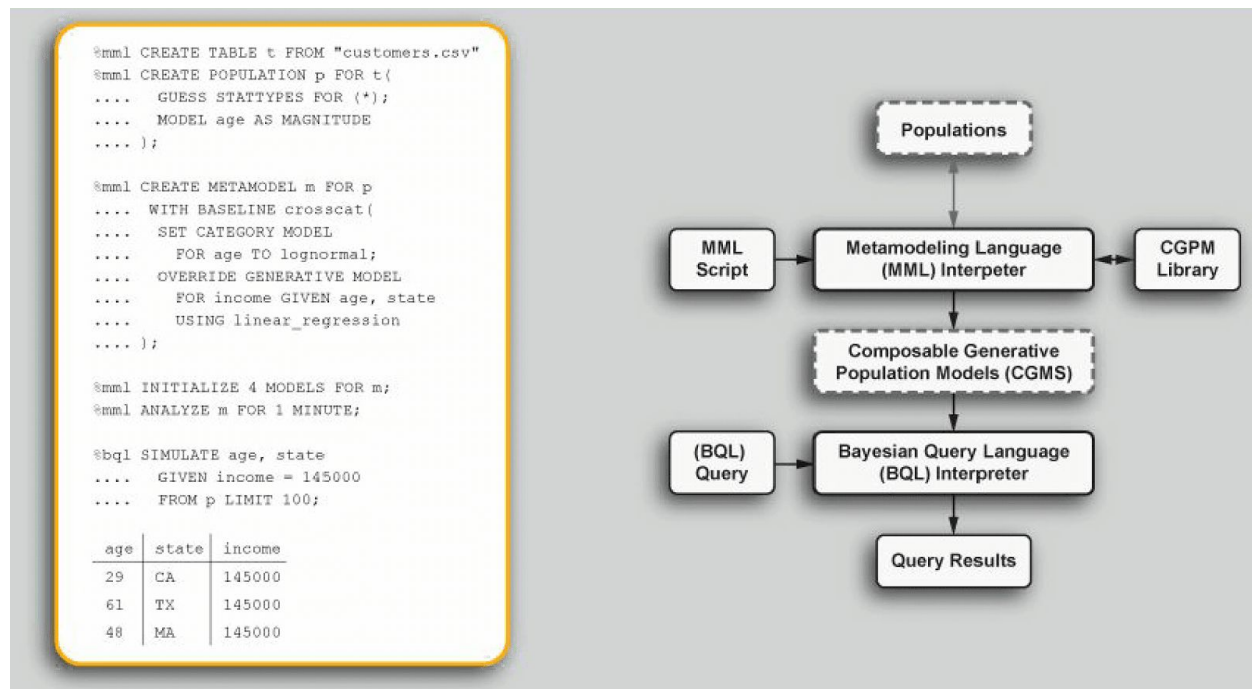


Figure 3. BayesDB for AI assistance for data science. The default modeling assumptions that BayesDB makes are suitable for a broad class of problems, but statisticians can customize these assumptions when necessary. BayesDB also enables domain experts that lack statistical expertise to perform qualitative model checking and encode simple forms of qualitative prior knowledge.

Picture

Picture is a probabilistic programming language for scene understanding that allows researchers to express complex generative vision models while automatically solving them using fast general-purpose inference machinery. Picture provides a stochastic scene language that can express generative models for arbitrary 2D/3D scenes, as well as a hierarchy of representation layers for comparing scene hypotheses with observed images by matching not simply pixels, but also more abstract features (e.g., contours, deep neural network activations). Inference can flexibly integrate advanced Monte Carlo strategies with fast bottom-up data-driven methods. Thus both representations and inference strategies can build directly on progress in discriminatively trained systems to make generative vision more robust and efficient.

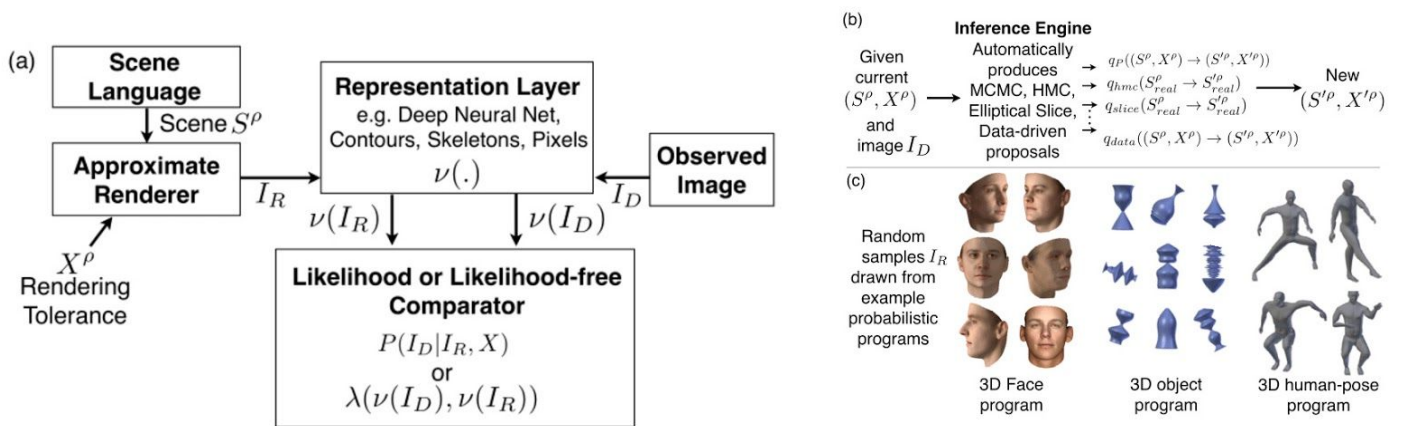


Figure 4. An overview of Picture, a probabilistic programming language for 3D visual scene understanding. (a) shows the architecture of Picture models. (b) shows the inference strategies used to fit models to data. (c) shows renderings of sampled 3D models from the priors used in three applications. Picture has been used to write ~50 line programs that solve problems in 3D face modeling, 3D human pose estimation, and 3D object reconstruction.

Venture

Venture is an extensible platform for probabilistic meta-programming in which probabilistic generative models, probability density functions, and probabilistic inference algorithms are all first-class objects. Any Venture program that makes random choices can be treated as a probabilistic model defined over the space of possible executions of the program. Such probabilistic model programs can also be run while recording the random choices that they make. Unlike other probabilistic programming platforms, Venture allows model programs, density meta-programs, and inference meta-programs to be written as user-space code in a single probabilistic programming language. Venture is essentially a Lisp-like higher-order language augmented with two novel abstractions: (i) probabilistic execution traces, a first-class

object that represents the sequence of random choices that a probabilistic program makes, and (ii) stochastic procedures, which encapsulate the probabilistic programs and meta-programs needed to allow simple probability distributions, user-space VentureScript programs, and foreign probabilistic programs to be treated uniformly as components of probabilistic computations. Venture also provides runtime support for stochastic regeneration of execution trace fragments that makes use of the programs and meta-programs of all stochastic procedures invoked during the execution of the original traced program.

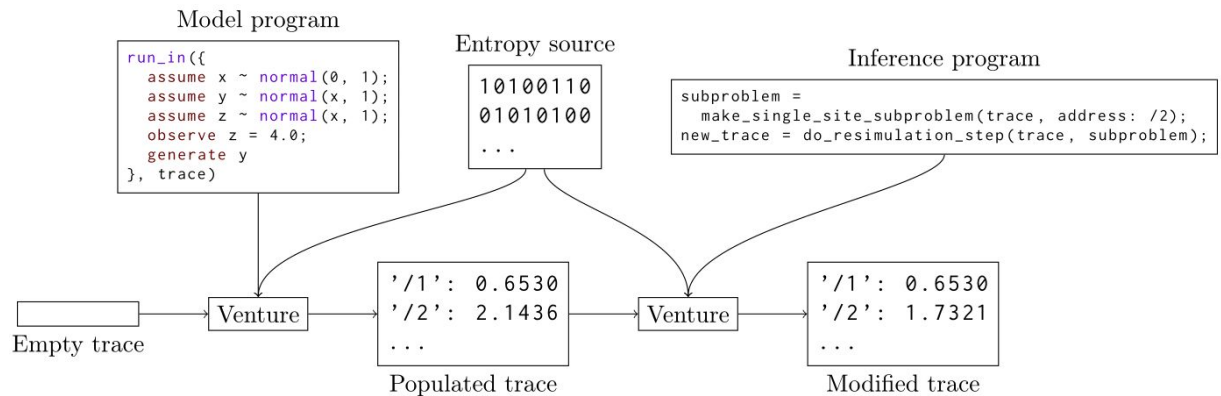


Figure 5. An overview of Venture. Users can specify models as programs, obtain traces of the stochastic choices made when these programs are run, and modify these traces using inference programs.

Goal 2: AI Capabilities

We demonstrated three knowledge-based AI capabilities:

1. Inferring the probable goals of autonomous agents
2. Inferring 3D models for visual scenes from single images
3. AI-assisted data science, based on probabilistic programs for inferring multivariate probabilistic models from empirical data



Figure 6. Example applications of probabilistic programming that were funded by our research on this program.

Capability #1: Probabilistic programs for inferring the probable goals of autonomous agents

Intelligent systems sometimes need to infer the probable goals of people, cars, and robots, based on partial observations of their motion. Our research introduces a class of probabilistic programs for formulating and solving these problems. The formulation uses randomized path planning algorithms as the basis for probabilistic models of the process by which autonomous agents plan to achieve their goals. Because these path planning algorithms do not have tractable likelihood functions, new inference algorithms are needed. Our research proposes two Monte Carlo techniques for these “likelihood-free” models, one of which can use likelihood

estimates from neural networks to accelerate inference. Our research demonstrates efficacy on three simple examples, each using under 50 lines of probabilistic code.

Building on these ideas, we believe it is now possible to write probabilistic programs that simultaneously capture several of the core common-sense reasoning and perceptual capacities used by humans from an early age (as young as 18 months) to interact helpfully and cooperatively with others, and to implement inference in these models with the efficiency needed to deploy them in robots that can interact cooperatively with humans in real time. The key idea is to organize common-sense knowledge in terms of a “probabilistic video game engine” that can (i) simulate interactions between objects and intentional agents, and (ii) judge the compatibility of a hypothesized scene with the available sense data. Fast, bottom-up inference for familiar objects, agents, and scenarios can be done via neural network proposals produced using standard Deep Learning techniques. Real-time inference for novel scenarios will be performed via massively parallel Monte Carlo algorithms. We have begun implementing this architecture using a new a high-performance probabilistic programming runtime system that we are developing, called Gen.jl, written in Julia to take advantage of modern many-core processors. Over the coming year, we hope to begin evaluating its predictions to behavioral studies of children and adults, and to use this architecture as the foundation for developing a range of autonomous system.

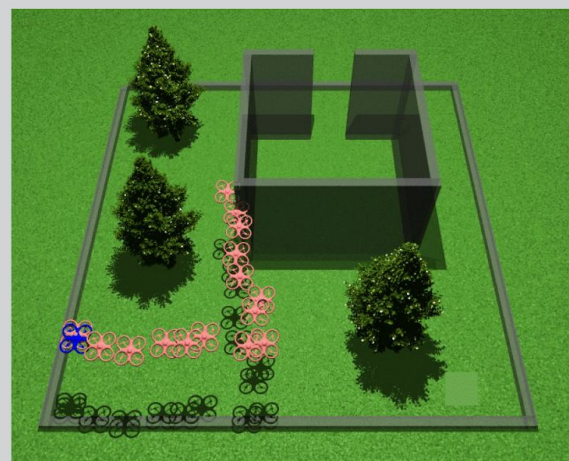
Inferring the destination of an autonomous agent

Problem: given noisy observations of motion, infer a drone's probable destination

Challenge: variety of environments and drone pilots; answers typically uncertain

Probabilistic programming approach:

1. build a probabilistic generative model using video-game components
2. do inference by combining Monte Carlo with deep learning on synthetic data



Starting location

Observed locations



Probable destinations

Figure 7. An overview of our probabilistic programming approach to inferring probable goals from observed agent motion.

Inferring the destination of an autonomous agent

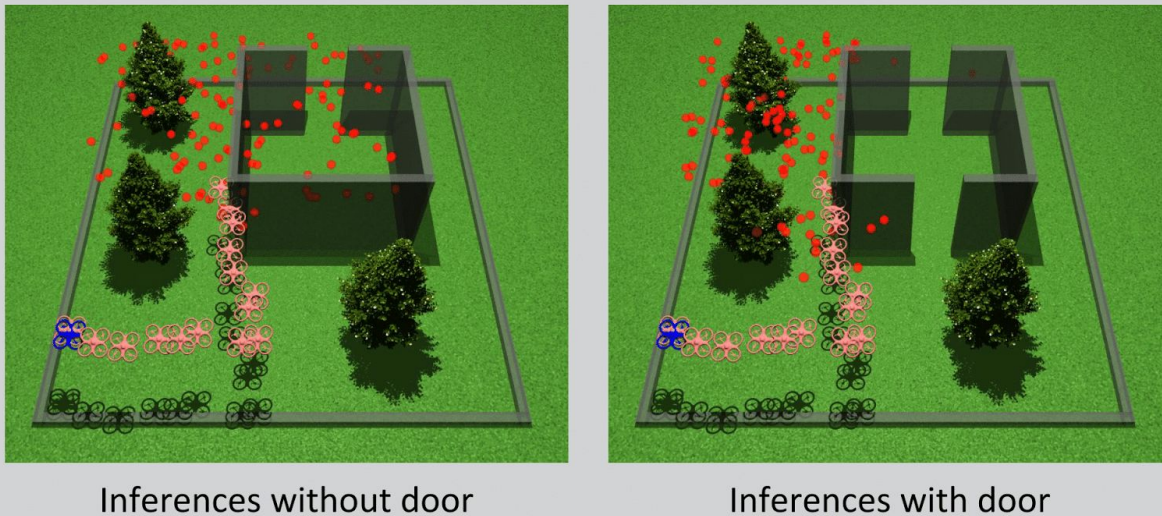


Figure 8. Goal inferences qualitatively cohere with human common-sense judgments. Small changes to the environment can produce large changes in inferences about probable goals.

In Scenario 1 (Figure 8, left), the drone's goal is more likely outside the enclosure, since it did not go directly into the enclosure through the bottom. In Scenario 2 (right), with bottom access to the enclosure, the goal is most likely outside the enclosure.

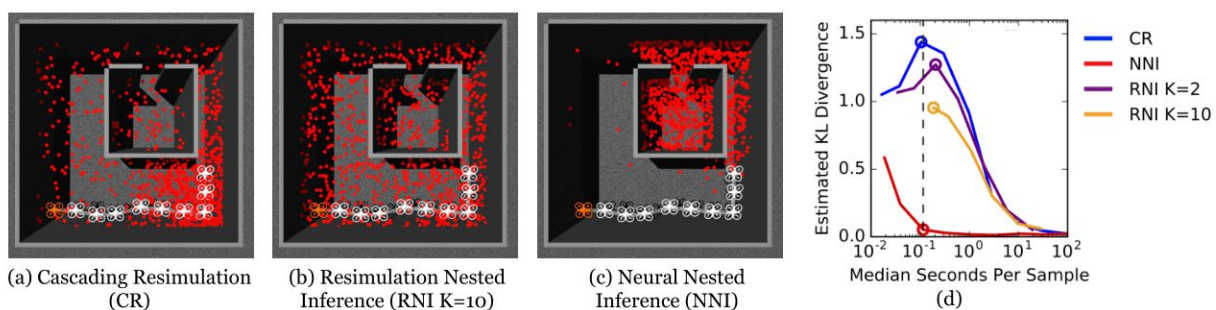


Figure 9. Comparison of three Metropolis-Hasting strategies for goal inference, which shows that neural Nested Inference MH converges faster than the other strategies.

(a), (b) and (c) show 960 independent approximate posterior goal samples (red) obtained using each strategy for similar run-times, given known map, start location (orange), and observations (white). Cascading Resimulation MH (CR) and Resimulation Nested Inference MH (RNI) do not give accurate inferences in real-time. Neural Nested Inference MH (NNI) uses a neural network

and gives accurate results in real-time (median 115 ms per sample). (d) shows estimated KL divergences from gold-standard samples to each of the strategies as the number of MH transitions are varied. Circles in (d) show the amount of computation used for (a,b,c).

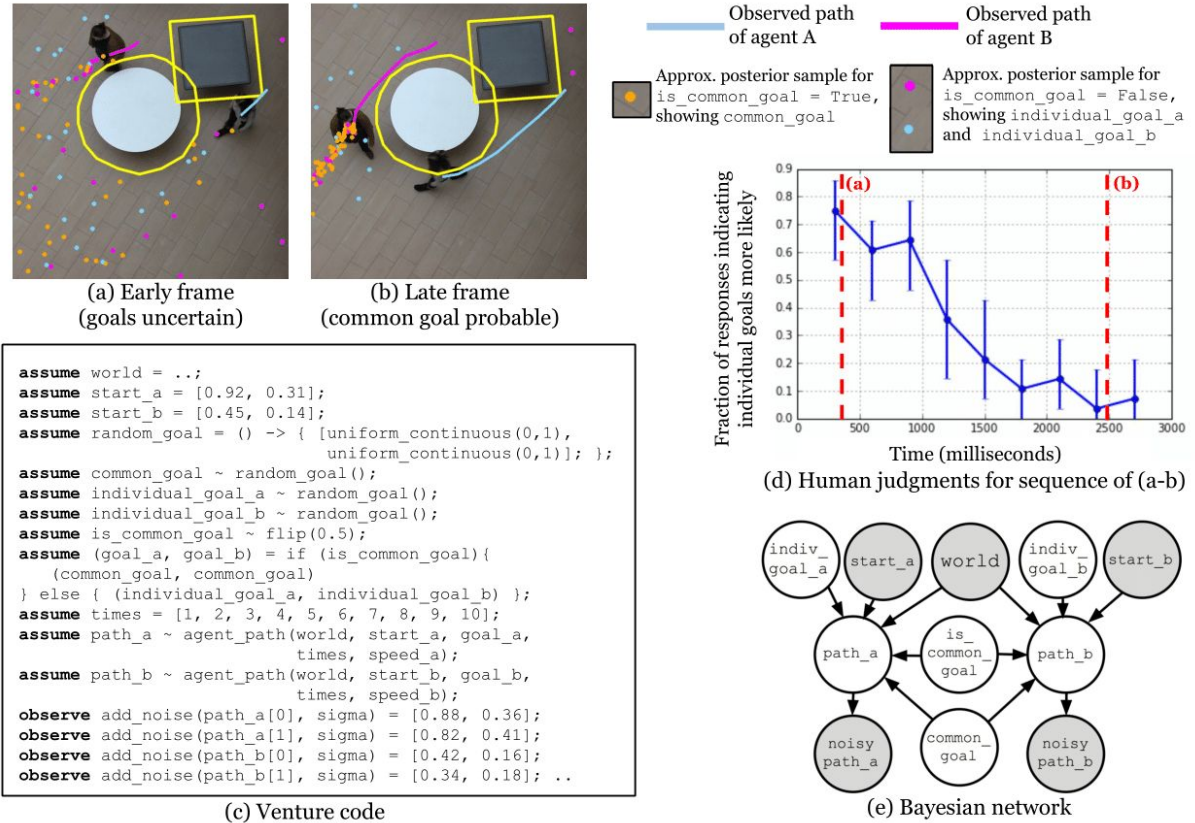


Figure 10. An overview of a Venture program for inferring whether or not two people are headed to the same destination. (a) and (b) show two frames from a movie, with goal inferences overlaid. (c) shows Venture source code for the probabilistic model. (d) shows judgments from 30 human responders of the likelihood over time that the individuals have different destinations, for the video sequence spanned by the frames in (a,b). The human judgments qualitatively agree with the automated inferences. (e) shows a Bayesian network schematic of the dependencies in the underlying probabilistic program from (c).

Capability #2: Probabilistic programs for inferring 3D scene structure from single images

Probabilistic scene understanding systems aim to produce high-probability descriptions of scenes conditioned on observed images or videos, typically either via discriminatively trained models or generative models in an “analysis by synthesis” framework. Discriminative approaches lend themselves to fast, bottom-up inference methods and relatively knowledge-free, data-intensive training regimes, and have been remarkably successful on many recognition problems. Generative approaches hold out the promise of analyzing complex scenes more richly and flexibly, but have been less widely embraced for two main reasons: Inference typically depends on slower forms of approximate inference, and both model-building and inference can involve considerable problem-specific engineering to obtain robust and reliable results. These factors make it difficult to develop simple variations on state-of-the-art models, to thoroughly explore the many possible combinations of modeling, representation, and inference strategies, or to richly integrate complementary discriminative and generative modeling approaches to the same problem. More generally, to handle increasingly realistic scenes, generative approaches will have to scale not just with respect to data size but also with respect to model and scene complexity. This scaling will arguably require general-purpose frameworks to compose, extend and automatically perform inference in complex structured generative models – tools that for the most part do not yet exist.

Picture is a probabilistic programming language that aims to provide a common representation language and inference engine suitable for a broad class of generative scene perception problems. We see probabilistic programming as key to realizing the promise of “vision as inverse graphics”. Generative models can be represented via stochastic code that samples hypothesized scenes and generates images given those scenes. Rich deterministic and stochastic data structures can express complex 3D scenes that are difficult to manually specify. Multiple representation and inference strategies are specifically designed to address the main perceived limitations of generative approaches to vision. Instead of requiring photo-realistic generative models with pixel-level matching to images, we can compare hypothesized scenes to observations using a hierarchy of more abstract image representations such as contours, discriminatively trained part-based skeletons, or deep neural network features. Available Markov Chain Monte Carlo (MCMC) inference algorithms include not only traditional Metropolis-Hastings, but also more advanced techniques for inference in high-dimensional continuous spaces, such as elliptical slice sampling, and Hamiltonian Monte Carlo which can exploit the gradients of automatically differentiable renderers. These top-down inference approaches are integrated with bottom-up and automatically constructed data-driven proposals, which can dramatically accelerate inference by eliminating most of the “burn in” time of traditional samplers and enabling rapid mode-switching.

"What does this face look like from the side? Or when lit differently?"

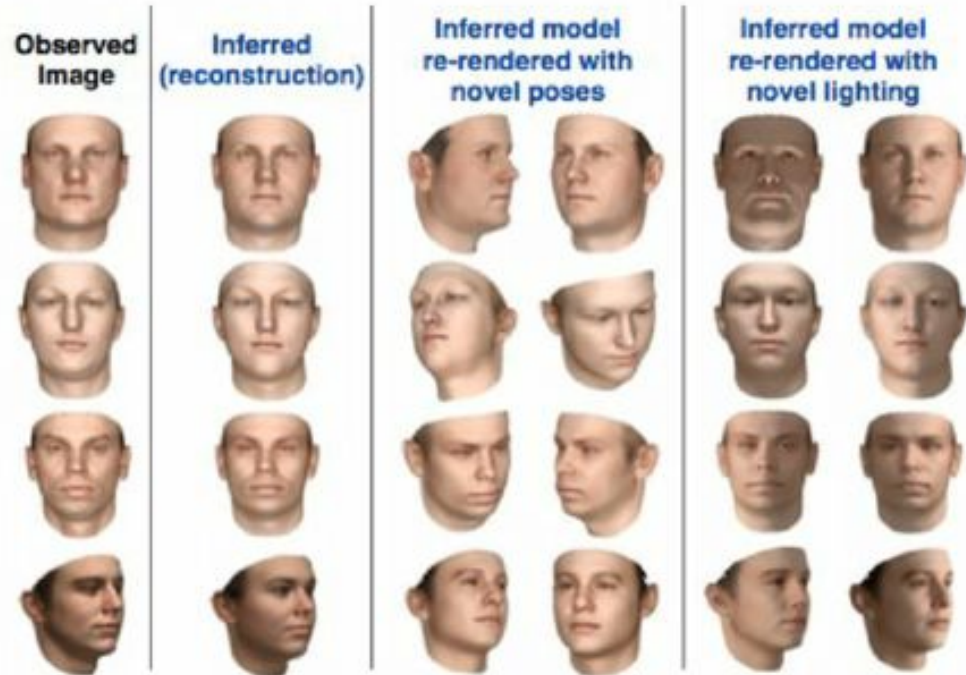


Figure 11. Picture inference on representative faces, answering the question "what does a given face probably look like when rotated or lit differently?". The first column shows the input images. The remaining columns show renderings of the inferred 3D models from a ~50-line probabilistic program written in Picture. This example shows that a short probabilistic program is applicable to non-frontal faces and provides reasonable parses, using only general-purpose inference machinery built into Picture.

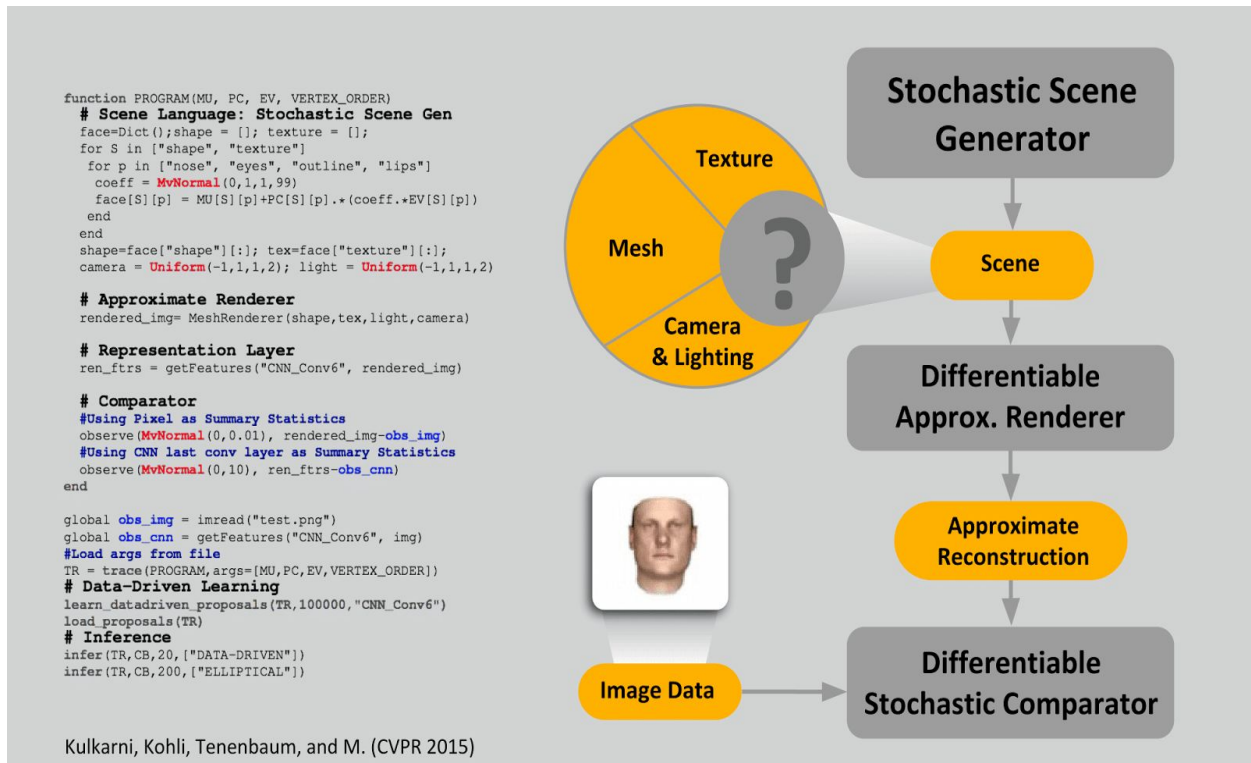


Figure 12. Picture code [left] for the 3D face application from Figure 11, along with a schematic description of the dependencies within that picture program [right].

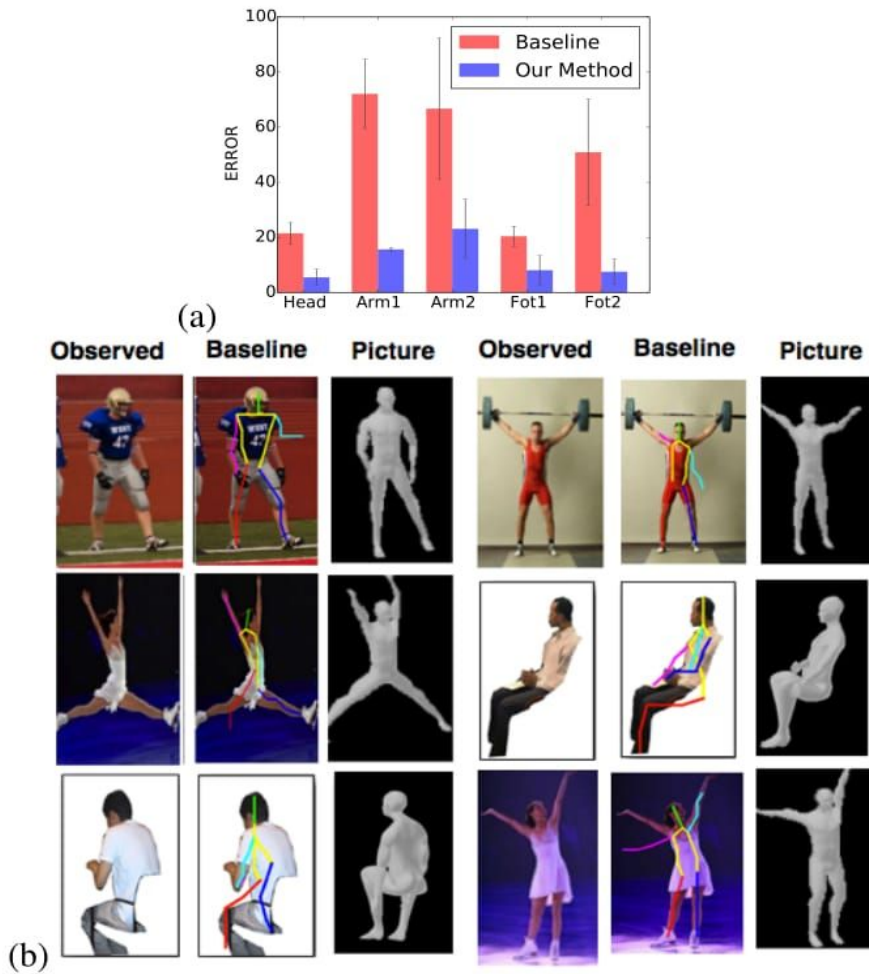


Figure 13. Quantitative and qualitative results for 3D human pose program. Our research on Picture quantitatively evaluated the pose program on a dataset collected from various sources, images with significant occlusion in the “person sitting” category, and the Internet. On the given dataset, as shown in the error histogram in (a), our model is more accurate on average than just using a DPM based human pose detector. The histogram shows average error for all methods considered over the entire dataset separated over the body part.

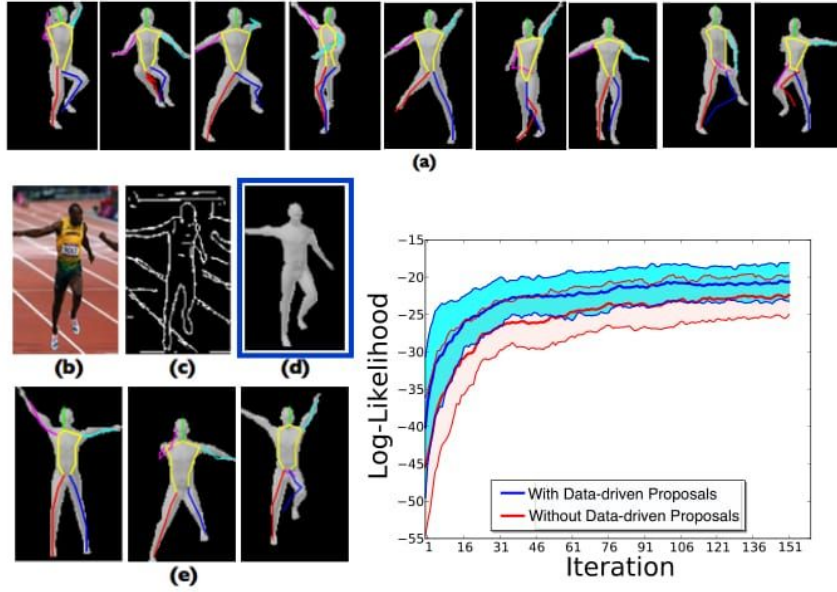


Figure 14. Data-driven proposal learning for 3D human pose program. (a) Random program traces sampled from the prior during training. The colored stick figures are the results of applying DPM pose model on the hallucinated data from the program. (b) Representative test image. (c) Visualization of the representation layer $v(l_D)$. (d) Result after inference. (e) Samples drawn from the learned bottom-up proposals conditioned on the test image are semantically close to the test image and results are fine-tuned by top-down inference to close the gap. As shown on the log-1 plot, we run about 100 independent chains with and without the learned proposal. Inference with a mixture kernel of learned bottom-up proposals and single-site MH consistently outperforms baseline in terms of both speed and accuracy.

For inference on faces, we obtained a 3D deformable face model trained on laser scanned faces. After training with this dataset, the model generates a mean shape mesh and mean texture map, along with principal components and eigenvectors. A new face can be rendered by randomly choosing coefficients for the 3D model and running the program. We evaluated the program on a held-out test set of 2D projected images of 3D laser scanned data. We additionally produced a dataset of about 30 images from the held-out set with different viewpoints and lighting conditions.

During experimentation, we discovered that since the number of latent variables is large (8 sets of 100 dimensional continuous coupled variables), elliptical slice moves are significantly more efficient than Metropolis-Hastings proposals. We also found that adding learned data-driven proposals significantly outperforms using only the elliptical slice proposals in terms of both speed and accuracy. We trained the data-driven proposals from around 100k program traces drawn from unconditional runs. The summary statistic function v_{dd} used were the top convolutional-layer features from the pre-trained ImageNet CNN model. The conditional

proposal density P_{density} was a multivariate kernel density function over cached latents with a Gaussian Kernel (0.01 bandwidth).

Many other academic researchers have used 3D deformable face models in an analysis-by-synthesis based approach. However, Picture is the only system to solve this as well as many other unrelated computer vision problems using a general-purpose system. Moreover, the data-driven proposals and abstract summary statistics (top convolutional-layer activations) allow us to tackle the problem without explicitly using 2D face landmarks as compared to traditional approaches.

We also developed a Picture program for parsing 3D poses of articulated humans from single images. Existing approaches typically require custom inference strategies and significant task-specific model engineering. In our probabilistic code, we use an existing base mesh of a human body, defined priors over bone location and joints, and enable the armature skin-modifier API via Picture's Blender engine API. The latent scene S_p in this program can be visualized as a tree with the root node around the center of the mesh, and consists of bone location variables, bone rotation variables and camera parameters. The representation layer v in this program uses fine-grained image contours and the comparator is expressed as the probabilistic chamfer distance.

We evaluated our program on a dataset of humans performing a variety of poses. This dataset was chosen to highlight the distinctive value of a graphics model-based approach, emphasizing certain dimensions of task difficulty while minimizing others: While graphics simulators for articulated bodies can represent arbitrarily complex body configurations, they are limited with respect to fine-grained appearance (e.g., skin and clothing), and fast methods for fine-grained contour detection currently work well only in low clutter environments.

We compared this approach with the discriminatively trained Deformable Parts Model (DPM) for pose estimation (referred as DPM-pose), which is notably a 2D pose model. As shown in Figure 13b, images with people sitting and heavy occlusion are very hard for the discriminative model to get right – mainly due to “missing” observation signal – while our model-based approach can handle these reasonably if we constrain the knee parameters to bend only in natural ways in the prior. Most of our model's failure cases, as shown in Figure 13b, are in inferring the arm position; this is typically due to noisy and low quality feature maps around the arm area due to its small size.

In order to quantitatively compare results, we project the 3D pose obtained from our model to 2D key-points. As shown in Figure 13a, our system localizes these key-points significantly better than DPM-pose on this dataset. However, DPM-pose is a much faster bottom-up method, and we explored ways to combine its strengths with our model-based approach, by using it as the basis for learning data-driven proposals.

Capability #3 - AI-assisted data science for time series

There is a widespread need for techniques that can discover structure from time series data. Recently introduced techniques such as Automatic Bayesian Covariance Discovery (ABCD) provide a way to find structure within a single time series by searching through a space of covariance kernels that is generated using a simple grammar. While ABCD can identify a broad class of temporal patterns, it is difficult to extend and can be brittle in practice. Our research shows how to extend ABCD by formulating it in terms of probabilistic program synthesis. The key technical ideas are to (i) represent models using abstract syntax trees for a domain-specific probabilistic language, and (ii) represent the time series model prior, likelihood, and search strategy using probabilistic programs in a sufficiently expressive language. The final probabilistic program is written in under 70 lines of probabilistic code in Venture. Our research demonstrates an application to time series clustering that involves a non-parametric extension to ABCD, experiments for interpolation and extrapolation on real-world econometric data, and improvements in accuracy over both non-parametric and standard regression baselines.

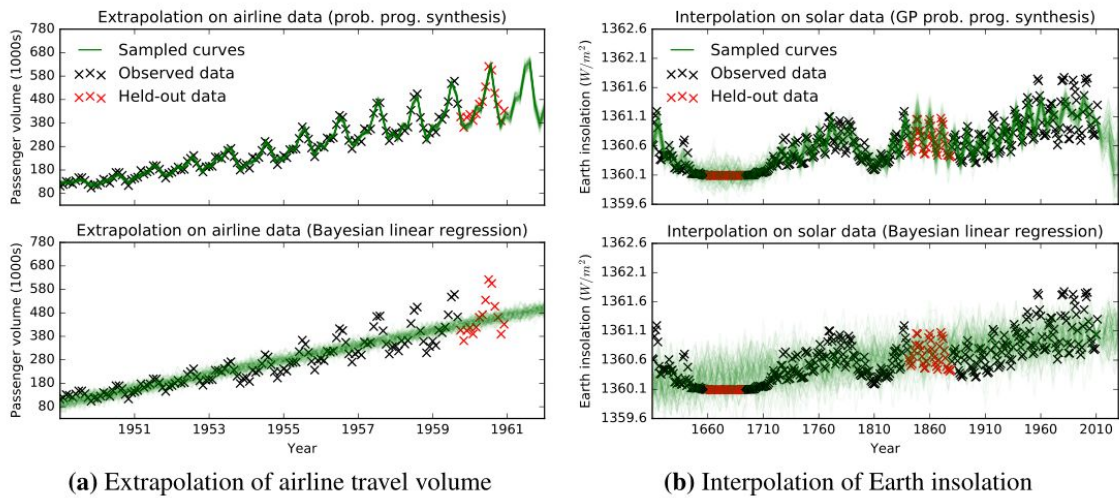


Figure 15. A comparison of our probabilistic program for AI-assisted time series extrapolation and interpolation (top) to regression modeling (bottom). The probabilistic program captures qualitative structure that regression modeling does not.

The figure above (a) shows extrapolation performance on a dataset of airline passenger volume between 1949 and 1960. The probabilistic program detects the linear trend with periodic variation, leading to very accurate predictions. (b) shows interpolation on a dataset of solar radiation between the years 1660 and 2010. The probabilistic program successfully models the qualitative change at around 1760, which correctly results in different interpolation characteristics at both ends. In contrast, Bayesian linear regression is forced to treat such

structural effects as unmodeled noise (Schaechtle, Saad, Radul, and Mansinghka, 2017).

	LOC for our probabilistic program	LOC for "Automated Statistician"
Main system	69	4,166
Gaussian process libraries	2,164	13,945
Generic inference implementation	1,887	–

Figure 16. Our probabilistic program implements functionality from the “Automated Statistician” but using ~100x fewer lines of code.

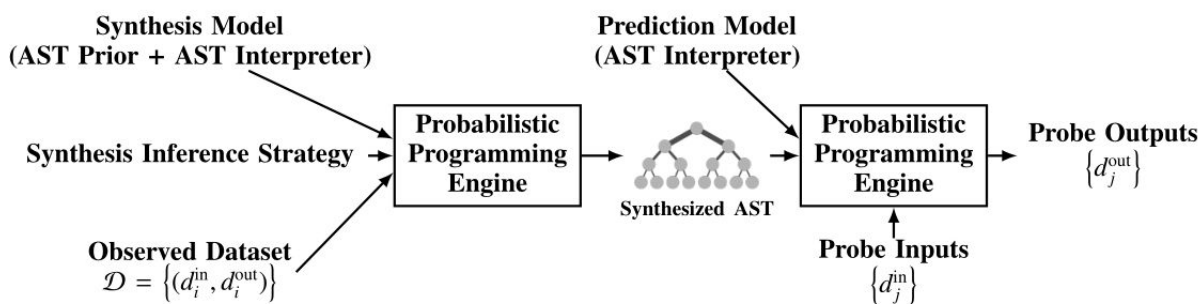


Figure 17. Overview of the architecture of our probabilistic program for AI-assisted time series analysis.

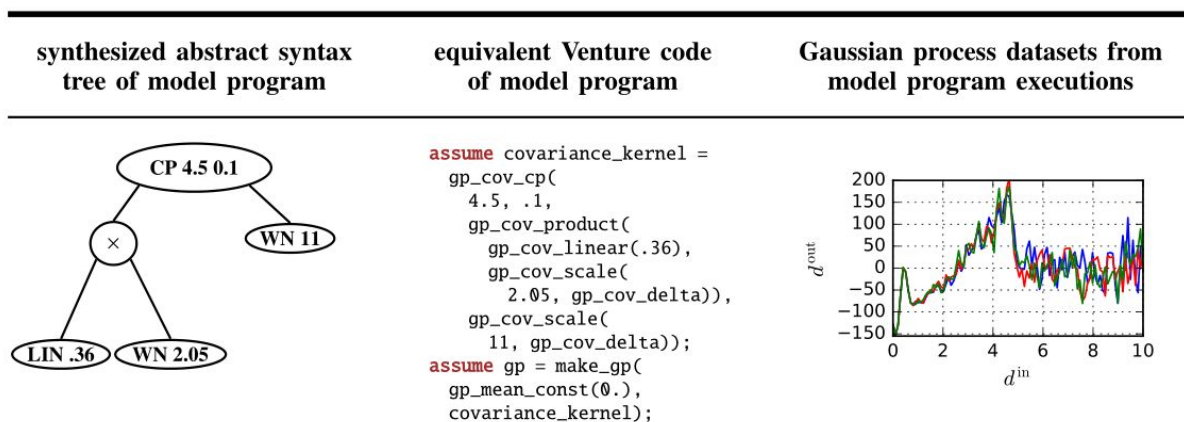


Figure 18. Example inputs and outputs from the internal components of our AI-assisted time series analysis program from Figure 17. Left: A symbolic structure generated by the AST prior. Center: Equivalent in Venture, produced by the AST interpreter. Right: Executions of the model program.

(a) Synthesis model: AST prior $\mathcal{G}$

```
1  assume tree_root = () -> {1};
2
3  assume get_hyper_prior ~ mem((node_index) -> {
4    // Gradient-safe exponential prior.
5    -log_logistic(log_odds_uniform() #hypers:node_index)
6  });
7
8  assume choose_primitive = (node) -> {
9    base_kernel ~ categorical(simplex(.2, .2, .2, .2, .2),
10     ["WN", "C", "LIN", "SE", "PER"]) #structure:pair("base_kernel", node);
11    cond(
12      (base_kernel == "WN") ([ "WN", get_hyper_prior(pair("WN", node))]),
13      (base_kernel == "C")  ([ "C", get_hyper_prior(pair("C", node))]),
14      (base_kernel == "LIN") ([ "LIN", get_hyper_prior(pair("LIN", node))]),
15      (base_kernel == "SE") ([ "SE", .01 + get_hyper_prior(pair("SE", node))]),
16      (base_kernel == "PER") ([ "PER",
17        .01 + get_hyper_prior(pair("PER_l", node)),
18        .01 + get_hyper_prior(pair("PER_t", node))
19      ])
20    );
21  };
22
23  assume choose_operator = mem((node) -> {
24    operator_symbol ~ categorical(simplex(0.45, 0.45, 0.1),
25     ["+", "*", "CP"]) #structure:pair("operator", node);
26    if (operator_symbol == "CP") {
27      [operator_symbol, get_hyper_prior(pair("CP", node))]
28    } else {
29      operator_symbol
30    }
31  });
32
33  assume generate_random_program = mem((node) -> {
34    if (flip(.3) #structure:pair("branch", node)) {
35      operator ~ choose_operator(node);
36      [operator, generate_random_program(2 * node), generate_random_program(2 * node + 1)]
37    } else {
38      choose_primitive(node)
39    }
40  });
```

(b) Synthesis model: AST interpreter $\mathcal{I}$

```
1 assume produce_covariance = (source) -> {
2   cond(
3     (source[0] == "WN") (gp_cov_scale(source[1], gp_cov_bump)),
4     (source[0] == "C") (gp_cov_const(source[1])),
5     (source[0] == "LIN") (gp_cov_linear(source[1])),
6     (source[0] == "SE") (gp_cov_se(source[1]**2)),
7     (source[0] == "PER") (gp_cov_periodic(source[1]**2, source[2])),
8     (source[0] == "+") (
9       gp_cov_sum(produce_covariance(source[1]), produce_covariance(source[2])),
10      (source[0] == "*") (
11        gp_cov_product(produce_covariance(source[1]), produce_covariance(source[2])),
12        (source[0][0] == "CP") (
13          gp_cov_cp(source[0][1], .1, produce_covariance(source[1]), produce_covariance(source[2]))
14        )
15      )
16    )
17  );
18 assume produce_executable = (source) -> {
19   baseline_noise = gp_cov_scale(.1, gp_cov_bump);
20   covariance_kernel = gp_cov_sum(produce_covariance(source), baseline_noise);
21   make_gp(gp_mean_const(0.), covariance_kernel)
22 };
```

(c) Data observation program

```
1 assume source ~ generate_random_program(tree_root());
2 assume gp_executable = produce_executable(source);
3 define xs = get_data_xs("./data.csv");
4 define ys = get_data_ys("./data.csv");
5 observe gp_executable({xs}) = ys;
```

(d) Synthesis inference strategy: MH + Gradients

```
for_each(arange(T), (_,) -> {
  infer gradient(
    minimal_subproblem(/?hypers), steps=100);
  infer resimulate(
    minimal_subproblem(one(/?structure)), steps=100))
```

Figure 19. Probabilistic program source code for AI-assisted time series analysis.

Capability #4: AI-assisted data science for databases

We have also developed BayesDB, a domain-specific probabilistic programming platform for probabilistic data analysis. BayesDB provides AI assistance for routine data analysis tasks, including data quality assessment, exploratory analysis, and predictive modeling. We have published 3 journal-length manuscripts (total ~150 pages) and 2 conference-length papers (~16 pages total) on BayesDB and applications.

BayesDB can be viewed as an AI platform for data science that aims to let domain experts solve problems in seconds or minutes that otherwise take hours or days for someone with good statistical judgment.

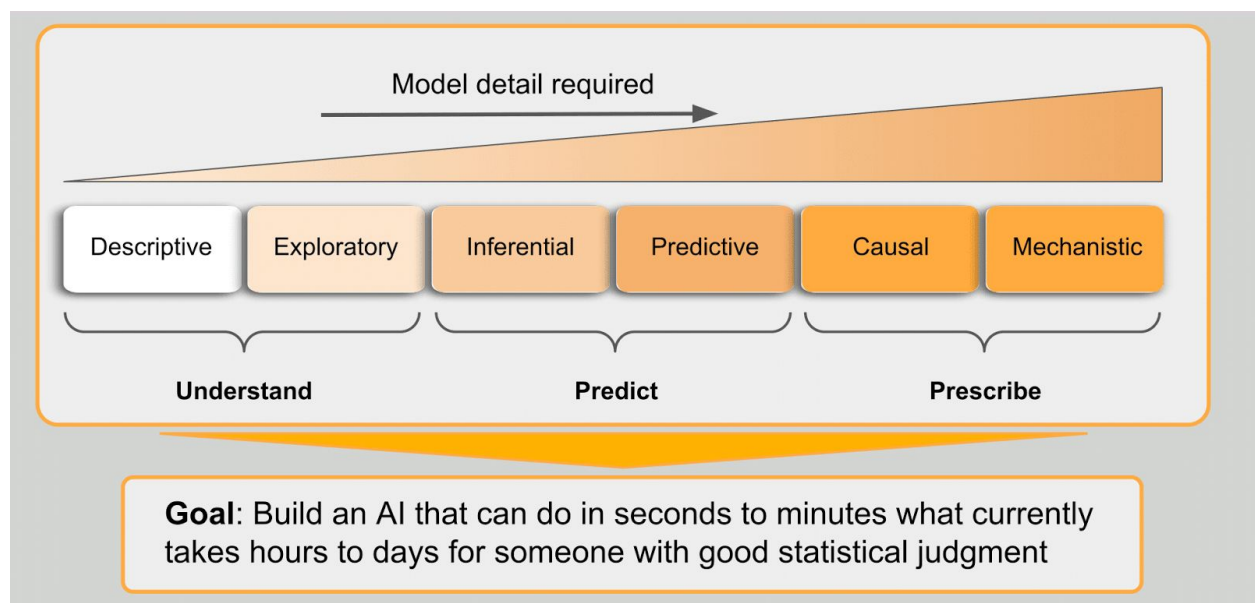


Figure 20. BayesDB provides AI assistance for data science.

AI is applicable to the entire spectrum of model understanding; an exploratory understanding of data can be done with AI only, but a mechanistic understanding would need to be done with AI and custom overrides.

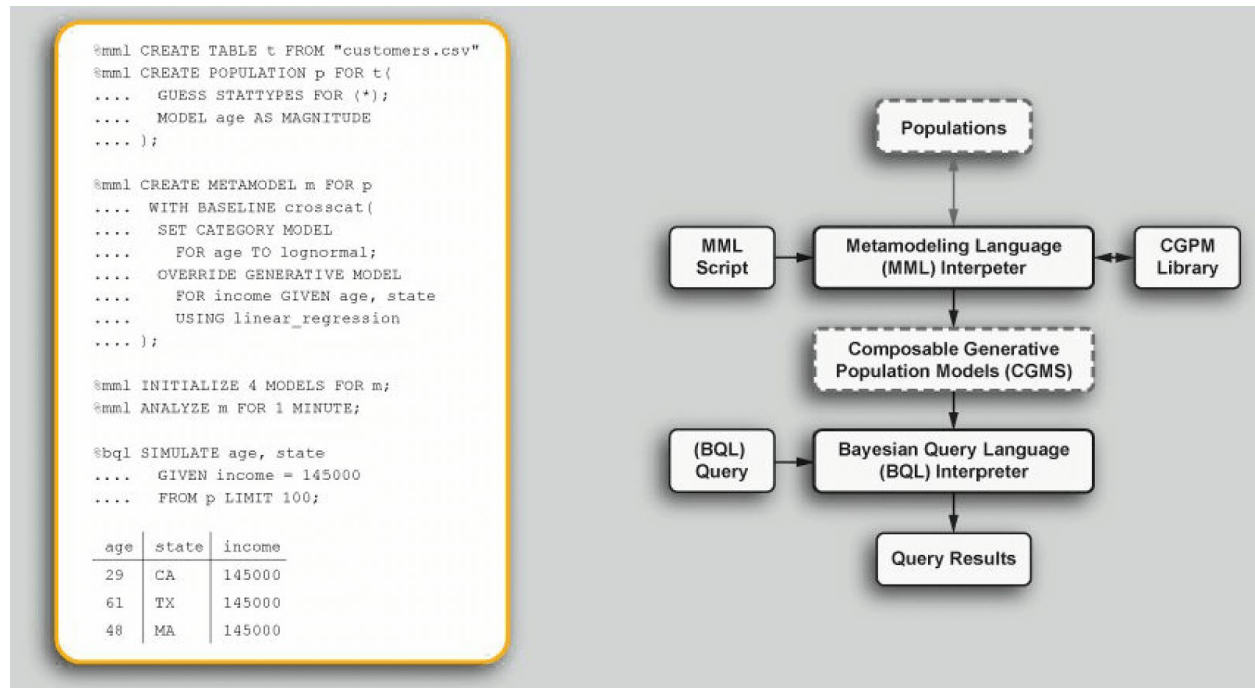
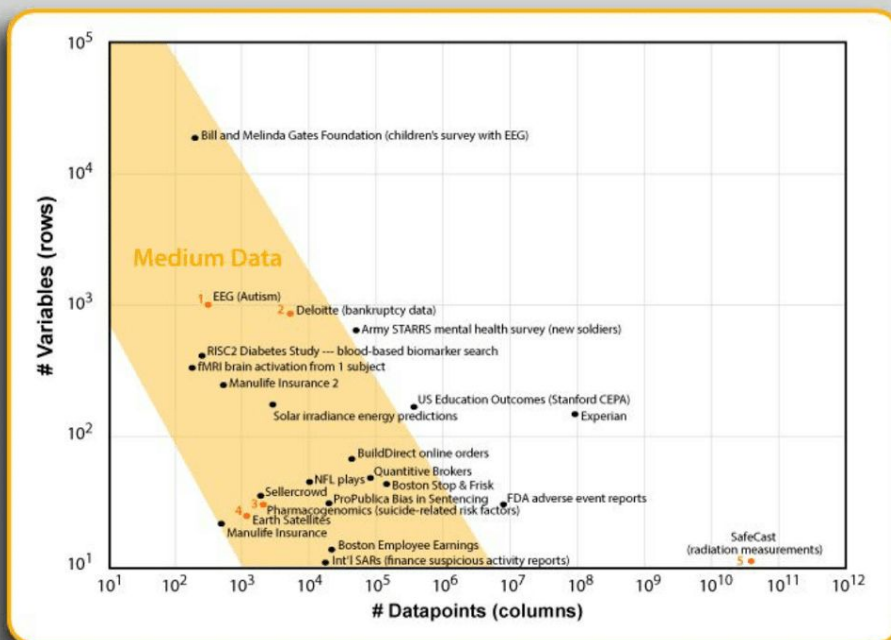


Figure 21. An example population construction in BayesDB [left] and an overview of the BayesDB architecture [right]. On the left, a population is constructed from a .CSV file; an ensemble of baseline models are built via automatic composition with CrossCat (reflecting some domain knowledge); and queried for predictions.



Questions

1. **EEG:** How is EEG different in an autistic child vs. a control?
2. **Deloitte:** Which companies are have similar bankruptcy risks?
3. **Pharmacogenomics:** Can we predict adverse events based on genetic data?
4. **Satellites:** What country probably owns an unidentified satellite?
5. **Safecast:** Which radioactivity measurements are unusual?

Figure 22. Examples of AI-assisted data science with BayesDB. We have applied BayesDB to a broad class of real-world databases, including extracts from the Allen Brain Atlas, the FDA Adverse Events database, and the Fragile Families database of at-risk families and children. Components of BayesDB have been the basis of two VC-backed startup companies; tutorials at the O'Reilly AI conference in NYC; and executive briefings at Intel. We are also engaged in early conversations surrounding transfer of BayesDB to DoD users via MIT Lincoln Laboratories, and to the intelligence community (IC) via conversations with IARPA leadership.

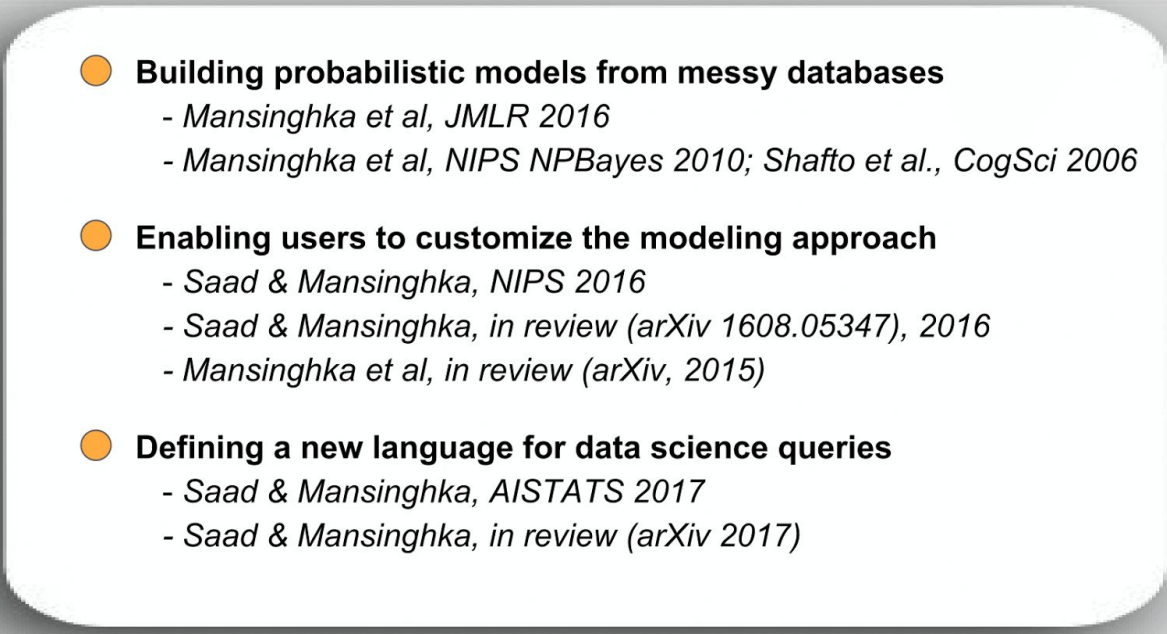
- 
- **Building probabilistic models from messy databases**
 - *Mansinghka et al, JMLR 2016*
 - *Mansinghka et al, NIPS NPBayes 2010; Shafto et al., CogSci 2006*
 - **Enabling users to customize the modeling approach**
 - *Saad & Mansinghka, NIPS 2016*
 - *Saad & Mansinghka, in review (arXiv 1608.05347), 2016*
 - *Mansinghka et al, in review (arXiv, 2015)*
 - **Defining a new language for data science queries**
 - *Saad & Mansinghka, AISTATS 2017*
 - *Saad & Mansinghka, in review (arXiv 2017)*

Figure 23. An overview of the main technical challenges involved in building BayesDB, and the associated publications that show how we addressed them.

Goal 3 - Fundamental Theory

Approximate probabilistic inference algorithms are central to many fields. Examples include sequential Monte Carlo inference in robotics, variational inference in machine learning, and Markov chain Monte Carlo inference in statistics. A key problem faced by practitioners is measuring the accuracy of an approximate inference algorithm on a specific dataset.

We have developed fundamental theory that addresses this problem. This theory leads to a new estimator for the accuracy of approximate inference algorithms called AIDE (short for "auxiliary inference divergence estimator"). AIDE is based on the observation that inference algorithms can be treated as probabilistic models and the random variables used within the inference algorithm can be viewed as auxiliary variables. This view leads to a new estimator for the symmetric KL divergence between the output distributions of two inference algorithms. Our research illustrates application of AIDE to algorithms for inference in regression, hidden Markov, and Dirichlet process mixture models. The experiments show that AIDE captures the qualitative behavior of a broad class of inference algorithms and can detect failure modes of inference algorithms that are missed by standard heuristics.

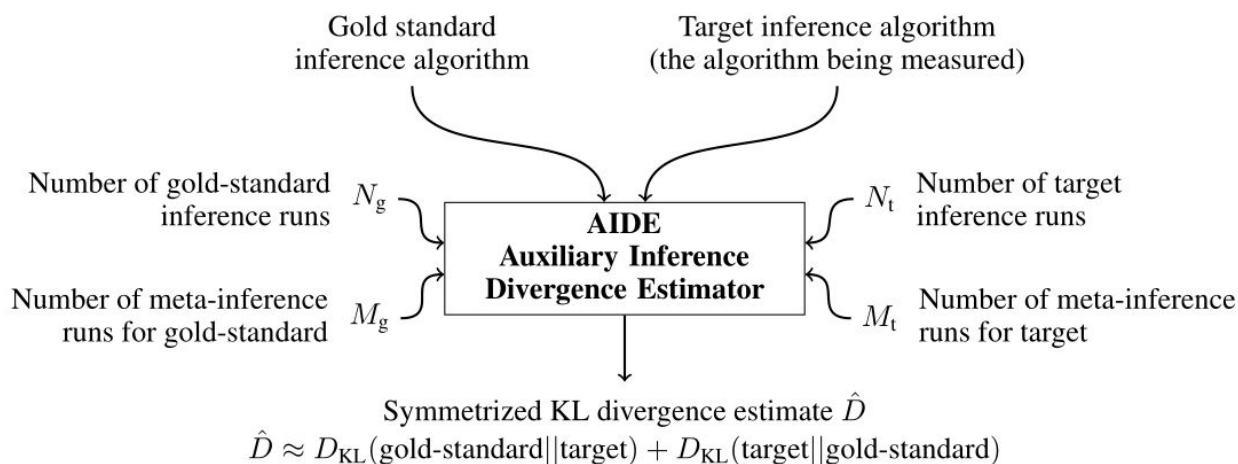


Figure 24. Using AIDE to estimate the accuracy of a target inference algorithm relative to a gold-standard inference algorithm. AIDE is a Monte Carlo estimator of the symmetrized Kullback-Leibler (KL) divergence between the output distributions of two inference algorithms. AIDE uses meta-inference: inference over the auxiliary random choices made by an inference algorithm.

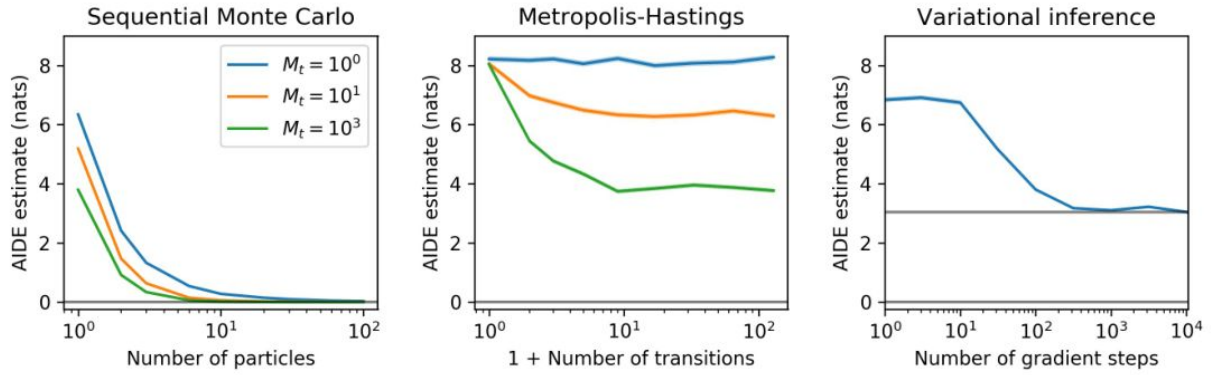


Figure 25. Comparing the bias of AIDE for different types of inference algorithms. Left: AIDE estimates for SMC converge to zero, as expected. Right: AIDE estimates for variational inference converge to a nonzero asymptote that depends on the variational family. Middle: The symmetrized divergence between MH and the posterior converges to zero, but AIDE over-estimates the divergence in expectation. Although increasing the number of meta-inference runs M_t reduces the bias of AIDE, AIDE is not yet practical for measuring MH accuracy due to inaccurate meta-inference for MH. The bias of AIDE is acceptable for SMC, and AIDE is unbiased for variational inference, but better MCMC meta-inference algorithms are needed to make AIDE practical for estimating the accuracy of MH.

Algorithm 2 Auxiliary Inference Divergence Estimator (AIDE)

Require: Gold-standard inference model and meta-inference algorithm $(\mathcal{U}, \mathcal{X}, q_g)$ and (r_g, ξ_g)
 Target inference model and meta-inference algorithm $(\mathcal{V}, \mathcal{X}, q_t)$ and (r_t, ξ_t)
 Number of runs of gold-standard algorithm N_g
 Number of runs of meta-inference sampler for gold-standard M_g
 Number of runs of target algorithm N_t
 Number of runs of meta-inference sampler for target M_t

for $n \leftarrow 1 \dots N_g$ **do**
 $u_{n,1}, x_n \sim q_g(u, x)$ $\triangleright$ Simulate from gold-standard generative inference model
 for $m \leftarrow 2 \dots M_g$ **do**
 $u_{n,m} \sim r_g(u; x_n)$ $\triangleright$ Run meta-inference sampler for gold-standard algorithm, on input x_n
 for $m \leftarrow 1 \dots M_t$ **do**
 $v_{n,m} \sim r_t(v; x_n)$ $\triangleright$ Run meta-inference sampler for target algorithm, on input x_n

for $n \leftarrow 1 \dots N_t$ **do**
 $v'_{n,1}, x'_n \sim q_t(v, x)$ $\triangleright$ Simulate from target generative inference model
 for $m \leftarrow 2 \dots M_t$ **do**
 $v'_{n,m} \sim r_t(v; x'_n)$ $\triangleright$ Run meta-inference sampler for target algorithm, on input x'_n
 for $m \leftarrow 1 \dots M_g$ **do**
 $u'_{n,m} \sim r_g(u; x'_n)$ $\triangleright$ Run meta-inference sampler for gold-standard algorithm, on input x'_n

$$\hat{D} \leftarrow \frac{1}{N_g} \sum_{n=1}^{N_g} \log \left(\frac{\frac{1}{M_g} \sum_{m=1}^{M_g} \xi_g(u_{n,m}, x_n)}{\frac{1}{M_t} \sum_{m=1}^{M_t} \xi_t(v_{n,m}, x_n)} \right) + \frac{1}{N_t} \sum_{n=1}^{N_t} \log \left(\frac{\frac{1}{M_t} \sum_{m=1}^{M_t} \xi_t(v'_{n,m}, x'_n)}{\frac{1}{M_g} \sum_{m=1}^{M_g} \xi_g(u'_{n,m}, x'_n)} \right)$$

return $\hat{D}$ $\triangleright \hat{D}$ is an estimate of $D_{\text{KL}}(q_g(x) || q_t(x)) + D_{\text{KL}}(q_t(x) || q_g(x))$

Figure 26. The algorithm for AIDE.

The generic AIDE algorithm above is defined in terms of abstract generative inference models and meta-inference algorithms.

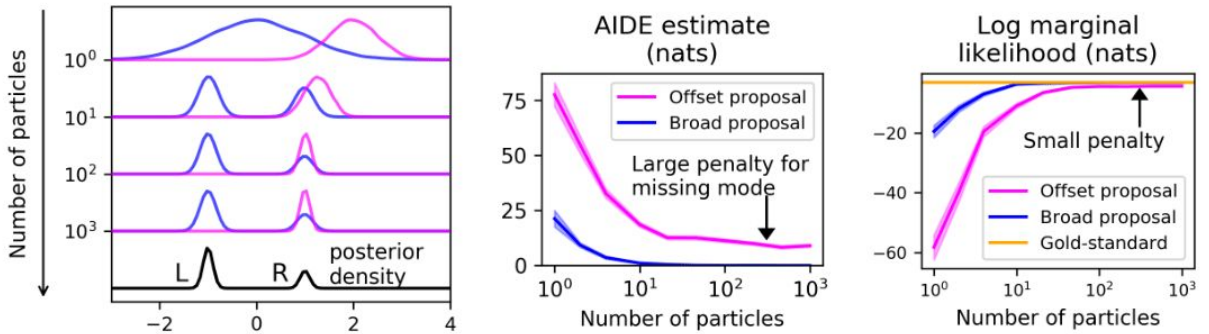


Figure 27. AIDE is able to detect when an approximate inference algorithm misses a posterior mode. Left: A bimodal posterior density, with kernel estimates of the output densities of importance sampling with resampling (SIR) using two proposals. The ‘broad’ proposal (blue) covers both modes, and the ‘offset’ proposal (pink) misses the ‘L’ mode. Middle: AIDE detects the missing mode in offset-proposal SIR. Right: Log marginal likelihood estimates suggest that the offset-proposal SIR is nearly converged.

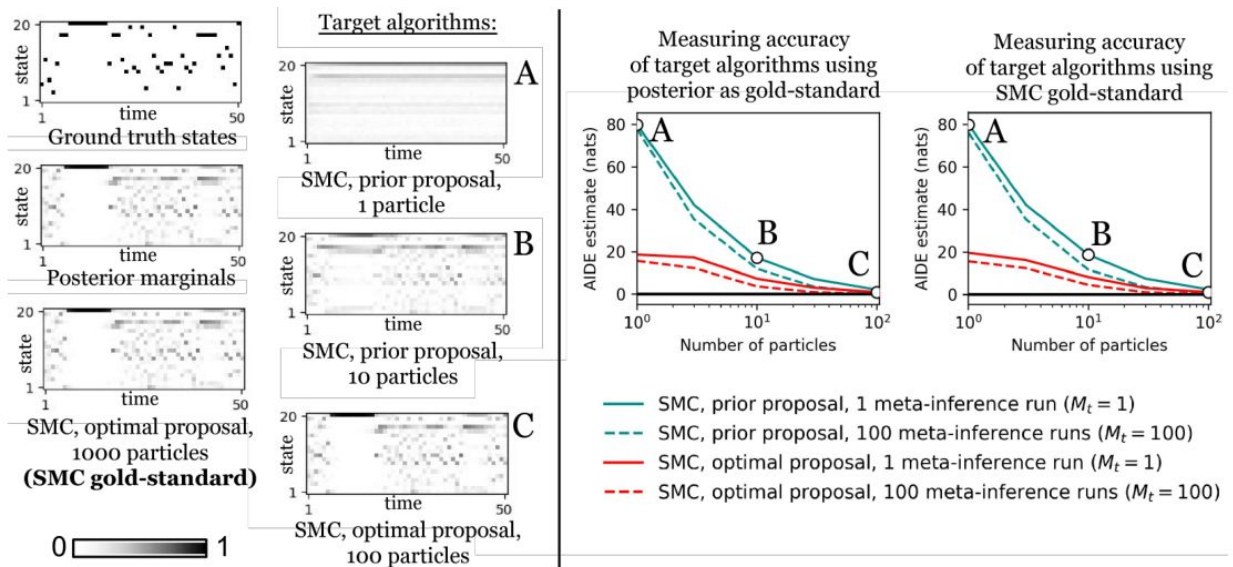


Figure 28. Comparison of exact posterior and ‘best-in-class’ approximate algorithm as gold-standard, when measuring accuracy of target inference algorithms with AIDE.

We consider inference in an HMM, so that exact posterior sampling is tractable using dynamic programming. Left: Ground truth latent states, posterior marginals, and marginals of the the output of a gold-standard and three target SMC algorithms (A,B,C) for a particular observation sequence. Right: AIDE estimates using the exact gold-standard and using the SMC gold-standard are nearly identical. The estimated divergence bounds decrease as the number of particles in the target sampler increases. The optimal proposal outperforms the prior proposal. Increasing M_t tightens the estimated divergence bounds. We used $M_g = 1$.

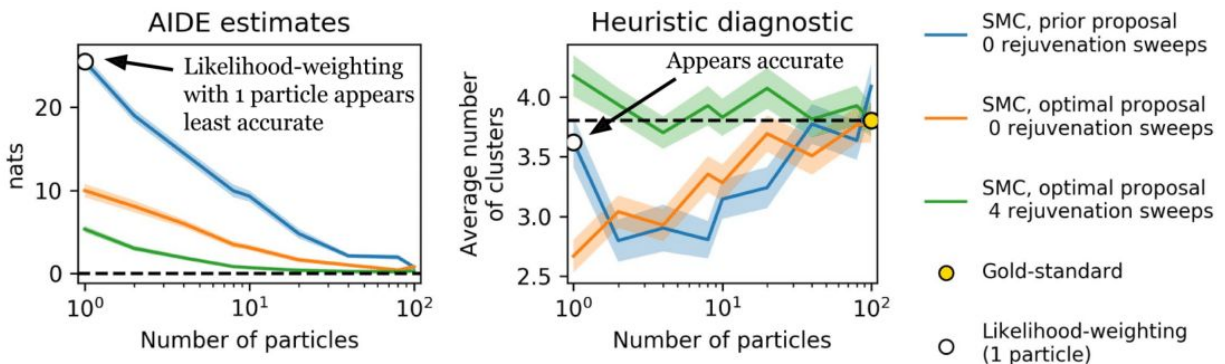


Figure 29. Contrasting AIDE against a heuristic convergence diagnostic for evaluating the accuracy of approximate inference in a Dirichlet process mixture model

The heuristic compares the expected number of clusters under the target approximation to the expectation under the gold-standard algorithm. White circles identify single-particle

likelihood-weighting, which samples from the prior. AIDE clearly indicates that single-particle likelihood-weighting is inaccurate, but the heuristic suggests it is accurate. Probe functions like the expected number of clusters can be error prone measures of convergence because they only track convergence along a specific projection of the distribution. In contrast, AIDE estimates a joint KL divergence. Shaded areas in both plots show the standard error. The amount of target inference computation used is the same for the two techniques, although AIDE performs a gold-standard meta-inference run for each target inference run.

Bibliography

1. Allen, K., Yildirim, I., and Tenenbaum, J. Integrating identification and perception: A case study of familiar and unfamiliar face processing. (2016).
2. Curlette, C., Schaechtle, U., and Mansinghka, V.K. Monitoring the Errors of Discriminative Models with Probabilistic Programming. Presented at the Workshop on Reliable Machine Learning in the Wild (NIPS - December 9, 2016), Barcelona, Spain.
3. Cusumano-Towner, M. and Mansinghka, V.K. AIDE: An algorithm for measuring the accuracy of probabilistic inference algorithms. *arXiv 1705.07224*, 2017.
4. Cusumano-Towner, M., Radul, A., Wingate, D., and Mansinghka V.K. Probabilistic programs for inferring the goals of autonomous agents. *arXiv 1704.04977*, 2017.
5. Feras, S., and Mansinghka, V.K. Probabilistic Data Analysis with Probabilistic Programming. *arXiv 1608.05347*, 2016.
6. Ghahramani, Z., Lloyd, J.R., Duvenaud, D., Grosse, R., and Tenenbaum, J. Building an Automated Statistician. UCL-Duke Workshop, September 2014.
7. Kulkarni, T.D., Kohli, P., Tenenbaum, J.B., and Mansinghka, V. Picture: A probabilistic programming language for scene perception. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR. pp. 4390-4399 (2015).
8. O'Toole, A. J., Edelman, S. & Bühlhoff, H. H. (1998). Stimulus-specific effects in face recognition over changes in viewpoint. *Vision Research*, 38, 2351-2363.
9. Richie, D. Quicksand: A Lightweight Embedding of Probabilistic Programming for Procedural Modeling and Design. The 3rd NIPS Workshop on Probabilistic Programming, 2014.
10. Saad, F., Casarsa, L., and Mansinghka, V. Probabilistic Search for Structured Data via Probabilistic Programming and Nonparametric Bayes. *arXiv 1704.01087*, 2017.
11. Saad, F. and Mansinghka, V. Probabilistic Data Analysis with Probabilistic Programming. *arXiv 1608.05347*, 2016.
12. Schaechtle, U. and Mansinghka, V. K. Gaussian Process Structure Learning via Probabilistic Inverse Compilation. *arXiv 1611.07051*, 2016.
13. Schaechtle, U., Saad, F., Radul, A., and Mansinghka, V.K. Time Series Structure Discovery via Probabilistic Program Synthesis. *arXiv 1611.07051*, 2017.