AFRL-RI-RS-TR-2022-035

BAYESDB FOR DATA-CENTRIC SCIENTIFIC DISCOVERY

MASSACHUSETTS INSTITUTE OF TECHNOLOGY (MIT)

FEBRUARY 2022

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-2022-035 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
CARL R. THOMAS
Work Unit Manager

/ S /
GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing and Communications 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
FEBRUARY 2022	FINAL TECHNICAL REPORT	SEPTEMBER 2017	SEPTEMBER 2021

4. TITLE AND SUBTITLE

BAYESDB FOR DATA-CENTRIC SCIENTIFIC DISCOVERY

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8750-17-C-0239	N/A	61101E
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
		R2DR

6. AUTHOR(S)

Vikash Mansinghka, Ulrich Schaecthle, Zane Shelby, Harish Tella, Cameron Freer, Feras Saad, Joshua Thayer, Amanda Brower, Rachael Paiste, Jonathan Rees, Alex Lew, Desiree Dudley, Marco Cusumano-Towner, Alexey Radul, Shivam Handa, Veronica Weiner, Joshua Tenenbaum

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Massachusetts Institute of Technology (MIT)
77 Massachusetts Ave
Cambridge MA 02139-4301

8. PERFORMING ORGANIZATION REPORT NUMBER

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/RITA 525 Brooks Road Rome NY 13441-4505	DARPA 675 North Randolph St Arlington VA 22203-2114	RI / DARPA	AFRL-RI-RS-TR-2022-035

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

This work was performed under an AFRL contract for developing data-driven methods to accelerate scientific discovery and robust design. Our MIT research team transitioned 4 results we believe constitute SD2 program-level accomplishments:

1. Automated data integrity via online learning of probabilistic programs, via Fail-Fast.

2. Fast exact symbolic inference for probabilistic programs, using SPPL.

3. Gen: A scalable, general-purpose probabilistic programming platform.

4. Learning whole-genome generative models from high-dimensional wet lab data to detect unwanted host-gene interactions and predict knockout effects on sample growth rate.

15. SUBJECT TERMS

Probabilistic programming, synthetic design, machine learning, whole genome sequencing, strain recommender

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	48

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
CARL R. THOMAS	N/A

Table of Contents

LIST OF FIGURES

List of Tables

Acknowledgements

We acknowledge Amin Espah Borujeni for Whole Genome Simulator work and for Figure 8; Enoch Yeung and Amin Espah Borujeni for helpful discussions around the strain recommender; Enoch Yeung and Dennis Manjaly Joshy for generating the evaluation data for the strain recommender; Ginkgo Bioworks for generating the RNA seq data; Biofab, Strateos and Ginkgo for flow cytometry experiments for early fail-fast work; Joshua Urrutia from TACC for the RNA seq pipeline he built; Matthew Vaughn and Eric Ferlanti from TACC for help testing the deployment of the SD2 Fail-Fast prototype; and the Institute for Protein Design at UW for ProStab data generation.

Ulrich Schaechtle was responsible for the majority of the applications research in this project, and made significant contributions to our original DARPA proposal. Schaechtle led multiple collaborations across teams within the SD2 program, starting from interdisciplinary problems that needed to be scoped via research and solved via a combination of research, engineering, and iterative field testing in close collaboration with end users. Schaechtle also provided the research engineering team leadership to build the probabilistic programming infrastructure that was used to carry out this work, guiding work by multiple software engineers and a UX researcher. In addition to delivering applications, this work helped drive several of the basic research advances that we describe in this report.

Veronica Weiner made significant contributions to our initial DARPA proposal, and as a member of the team for the first three months, carried out our team's first tests on the ProStab problem.

Rachel Paiste provided multiple years of program management that made it possible for our lab to align multiple basic and applied research streams. Paiste also helped us identify key weaknesses in our visualizations; fixing them substantially changed the framing of our research and increased comprehension by synthetic biologists.

Amanda Brower provided fiscal support and lab management throughout the program, making it operationally possible for us to carry out this research. Desiree Dudley also provided program management support, helping Schaechtle and Mansinghka navigate alarge transition in research priorities.

Feras Saad, a PhD student at MIT ProbComp, took significant basic research risks in addressing the scaling challenges with probabilistic programming that we encountered atthe start of the program.

1.0 SUMMARY

The Synergistic Discovery and Design (SD2) program set out to “develop data-driven methods to accelerate scientific discovery and robust design in domainsthat lack complete models.” Our MIT research team has successfully demonstrated
data-driven solutions to multiple hard discovery & design problems in synthetic biology, leveraging breakthrough probabilistic programming research results developed over thecourse of the SD2 program. We have also developed open-source prototypes implementing some of these research results that have attracted millions of additional dollars of transition funding, from both industry (via Takeda Pharmaceuticals, IBM, and Intel) and DARPA (via the MCS and SAIL-ON programs).

The central scientific advance enabling our success is new AI technology that learns grounded, data-driven probabilistic programs from real-world wet lab experiments. These probabilistic programs can generate synthetic values for thousandsof measurements and tens of experimental and design variables, screen new experiments for errors, and classify experimental samples into empirically-discovered "operating regimes". In some cases, they can act like a "virtual wet lab", filling an analogous role to classical computational modeling techniques, but apply to data-drivendomains where even qualitative causal knowledge is not available. These probabilistic programs can also be analyzed for causal information about the biological system understudy. Furthermore, many of these capabilities are accessible via a simple, SQL-like language that is intended to be learnable by biologists (and experts in other domains) who lack an understanding of data science.

This level of AI assistance for discovery and design could potentially have long-term, transformative impact across multiple STEM fields. Our SD2 results primarilydemonstrated efficacy for synthetic biology, and our transitions substantiate value for medical research. Clinical trials and clinical research are an especially important application area; reducing the data and time needed for clinical discoveries could save lives and reduce unnecessary human suffering. We also believe that, longer term, SD2's core capabilities around data-centric discovery could open up new opportunities for the DoD and federal government in policy and in operational decision support. AI assisted model discovery and decision support from sparse administrative and operational data could add value by improving the quality of data, by surfacing relationships that might not be apparent to human experts, and by providing automated,data-driven decision support.

Our probabilistic programming approach overcomes multiple technical limitationsof machine learning approaches on these problems. Probabilistic programs can be learned accurately from "small and wide" data --- i.e. data where the number of columns/variables/dimensions vastly outstrips the number of rows/datapoints/experimental replicates --- via Bayesian structure learning techniques. In this regime, high-capacity machine learning techniques (e.g. deep learning) can easily overfit, yet standard statistical techniques (e.g. linear modeling, widely used by

biologists) can miss significant non-linear and/or heteroscedastic effects and thereby cause design failure. Probabilistic programming "knows when it doesn't know", by quantifying uncertainty over model structure and over all inference results. It smoothlyincorporates expert knowledge when available (via manual edits to the human-interpretable probabilistic program source code produced by our learning mechanisms). Domain experts, e.g. biologists, can pose new questions themselves andget answers to challenging statistical inference and data science problems without the assistance of statisticians / machine learning experts. Probabilistic programs can automatically detect dataset drift and data entry errors, and sometimes even detect wetlab protocol violations. They are also fast and deployable in web browsers, and thus a suitable basis for interactive applications that can assist scientists and engineers in discovery and design.

While carrying out this research, our team achieved and transitioned four significantresults that we believe constitute significant SD2 program-level accomplishments:

1. Learning whole-genome generative models from high-dimensional wet lab data
2. Fail-Fast: automated data integrity via online learning of probabilistic programs
3. Fast exact symbolic inference for probabilistic programs
4. Gen: a scalable, general-purpose probabilistic programming platform

Funded transitions to other DARPA and industry research programs are underway:

- To the DARPA SAIL-ON program, for accomplishments #3 and #4
- To the DARPA MCS program, for accomplishment #4
- To Takeda Pharmaceuticals, for accomplishments #2 and #3
- To Intel and IBM, for accomplishment #4

1. Learning whole-genome generative models from high-dimensional wet lab data

When we began, wet lab experiments in synthetic biology produced high-dimensional data (e.g. ~4000 RNAseq measurements per set of experimental conditions) where the number of variables far outstrips the number of experiments (~100-1000 experiments). Analyzing this data is time-consuming and error-prone.

We created new algorithms for learning whole-genome generative models that simultaneously simulate all ~4000 genes, using explainable models whose structure canbe analyzed and queried in real time. The metrics show these models successfully detect unwanted host-gene interactions and predict knockout effects on sample growth rate, on SD2 Novel Chassis data. Use cases in causal inference --- e.g. extracting interpretable causal relationships from the source code of whole-genome simulators --- may also be possible with further research.

This technology is ready for research use on novel organisms, and is beingtested operationally by biologists from the TA2 Voigt Lab.

2. Fail-Fast: automated data integrity via online learning of probabilistic programs

When we began, people collecting data from wet labs had no way to detect critical dataand protocol errors before the last stages of data analysis. This led to wasted time and money, and slowed the pace of discovery and design. A key goal of SD2 was to developdata-driven technology that respected domain knowledge and leveraged expected coherence of results across multiple labs, to detect errors and anomalies

early in the process, and therefore reduce the time and cost of discovery and design.

We created SD2 Fail-Fast, an open-source system that automatically screens multivariate data from complex real-world experiments to detect probable data errors, protocol violations, and anomalous results. SD2 Fail-Fast detects deeper semantic errors than simple deterministic rules, by automatically learning explainable, symbolic probabilistic programs that model multivariate data streams (leveraging domain knowledge) and using them to infer probable errors, protocol violations, and anomalousvalues. The metrics show SD2 Fail-Fast detected data errors in SD2 ProStab and otherearly SD2 challenge problems.

Takeda Pharmaceuticals values SD2 Fail-Fast's technology (and the underlying probabilistic programming platform used to develop Fail-Fast) and has funded its furtherdevelopment. It allows Takeda to automatically detect probable errors from real clinical trials, and helps identify actionable interventions (e.g. carrying out a monitoring visit for a specific field site that is generating a high rate of anomalous data). This approach could potentially help Takeda save millions of dollars and reduce the time-to-market for new treatments.

Large-scale operational deployment requires commercialization (e.g. via venturecapital) and potentially also development of an open-source community. The main technical roadblocks are in making this technology easier to deploy for new applications, and in quantifying ROI for domains such as clinical trials.

3. Fast exact symbolic inference for probabilistic programs

When we began, probabilistic programs were widely believed to be too slow andinaccurate for many practical applications. The key computational bottleneck is inference: taking a probabilistic program and using it to generate simulations of or calculating probabilities of specific (and potentially rare) events of interest.

We created SPPL, a new probabilistic programming system that delivers exact probabilities and simulations, ~1000x faster results than the previous state of the art. SPPL makes it practical to calculate probabilities of rare events in milliseconds, and useprobabilistic inference in real-time interfaces, e.g. by biologists using SPPL programs toscreen designs for impact on host organisms.

The metrics show that SPPL makes it practical to automatically analyze whole-genome simulator source code to detect probable host effects (whereas it was prohibitively slow beforehand). Large-scale operational testing is feasible today, using open-source SD2 prototypes, though the software needs to be hardened, documented,and maintained by professional engineers before operational use.

4. Gen: a scalable, general-purpose probabilistic programming platform

When we began, probabilistic programs were only applied in narrow domains such as Bayesian statistics and data analysis. Inference was not customizable enoughor fast enough for many real-world applications.

We created Gen, a new probabilistic programming platform that makes it practical to apply probabilistic inference to solve hard problems in systems biology and computer vision, among other domains. The metrics show that Gen programs (i) can require ~20x fewer lines of code than typical expert implementations; (ii) support hybridsof neural, symbolic, and Monte Carlo inference that deliver state of the art (SOTA) accuracy; and (iii) are ~1000x faster than previous probabilistic programming languageswith custom inference.

Our Intel, IBM, and DARPA MCS transition partners value Gen's unique ability todeliver real-time inference of uncertain structure from real-world data --- not just

estimate parameters --- and to enable new hybrids of neural, symbolic, and probabilisticAI techniques. Intel, IBM, and DARPA MCS have provided follow-on funding, and are using Gen as the basis for fundamental new research in building AI with human-like common-sense.

Users at UW Madison have independently applied Gen to learn causal network structures of signaling pathways from phosphorylation time series data to better understand cell regulation, opening up new approaches to fundamental data-driven modeling challenges that are relevant for discovery and design [1]. A second group of researchers outside of our lab, from MIT and Northeastern, independently used Gen fornetwork-based epidemiological simulations [2].

Large-scale research use is currently underway at multiple US and international universities. Operational use is possible but not advisable until industry and academia and government create an open-source ecosystem that provides the necessary maintenance and community support. It is encouraging that both Google and Microsoft have begun exploring large-scale investments in probabilistic programming, to grow thisopen source project, on the strength of results that were first funded by DARPA PPAMLand then SD2.

2. INTRODUCTION

In this section we briefly review the technical approach behind our research. We focus on three aspects:

1. Integration of diverse experimental data sources into a standardized data tableformat. Once in this format, our systems learn probabilistic programs that can help solve a broad class of discovery and design problems. (Figure 1)

2. Use of our probabilistic programs for (i) predictive modeling of the host impact ofnovel genes; and (ii) recommending strains for engineering. Other sections of this report describe (iii) detecting data quality issues. (Figure 2)

3. Probabilistic programming, the AI approach enabling the capabilities in #2 above.

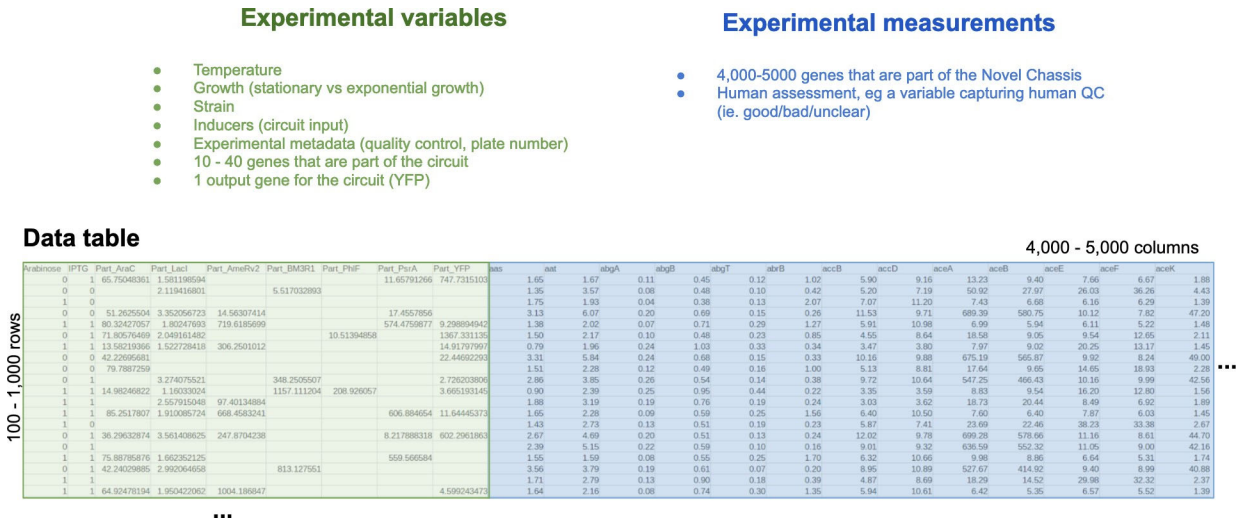


Figure 1. A high-level overview of the typical data assembly approach we used, combining ~10s of experimental variables with ~4000-5000 experimental measurements.

Figure 1: This "small and wide" data is challenging to analyze via traditional high-dimensional statistics techniques, as the number of rows (experimental samples) is far outstripped by the number of variables.

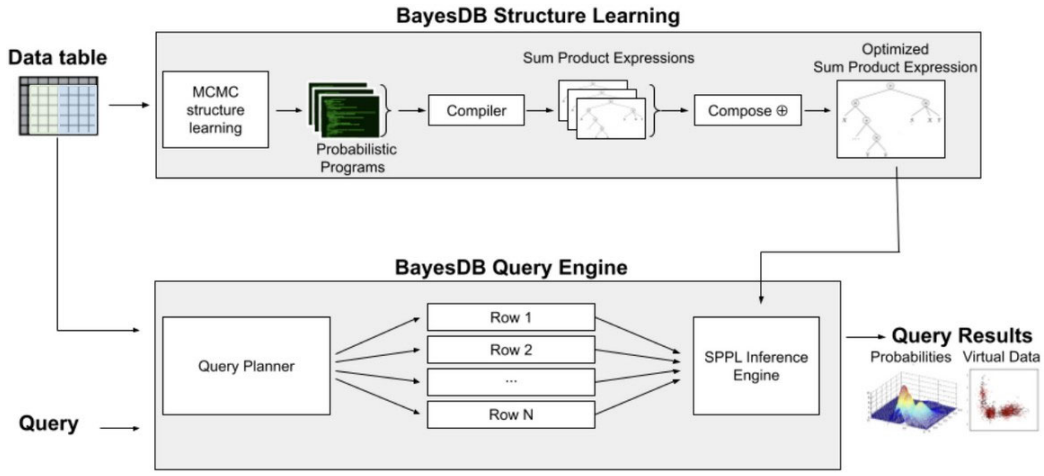


Figure 2..The new BayesDB system accomplishes one of the core program goals for SD2: it implements generalizable, automated infrastructure for scientific data analysis.

Figure 2: The system consists of two components that allow users to move from data and queries to results: (i)the structure learning component and (ii) the query engine. (i) takes data and generates a sum product expression. The system then takes that expression, along with raw data, and uses themboth to inform the answers it provides to users of (ii). In the SD2 use case, BayesDB generates virtual experiment results and assesses interactions between variables of interest such as host genes or circuit interactions.

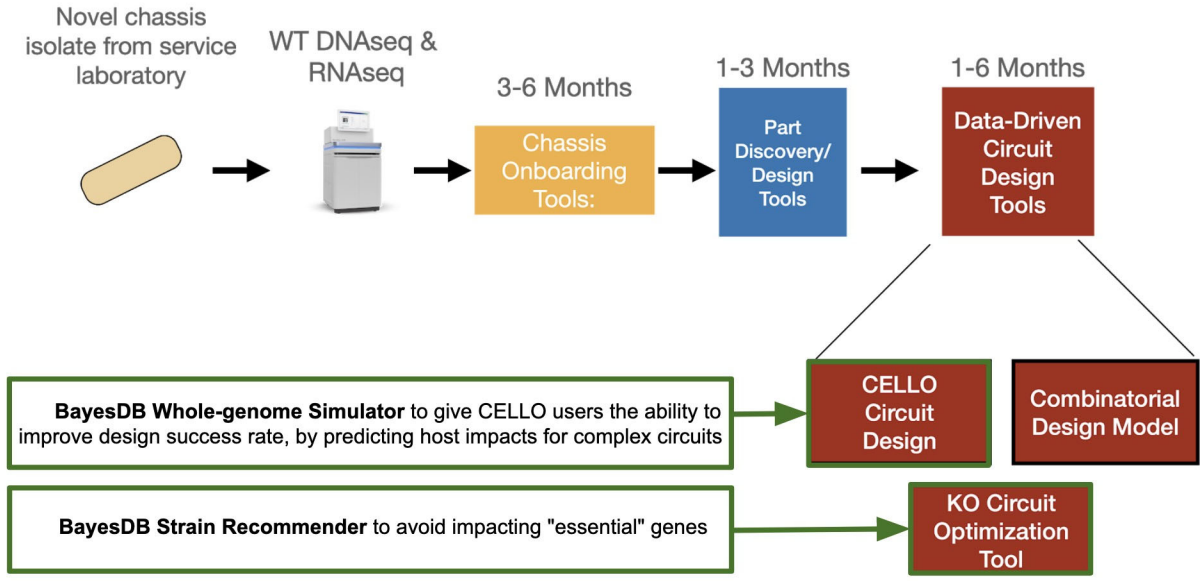


Figure 3. Figure 3. How the BayesDB Whole-genome Simulator and BayesDB Strain Recommender fit into the Novel Chassis ecosystem.

Figure 3: We worked with the Voigt Lab and with UCSB to build, debug, and experimentally validate both of these systems. Section 3 of this report describes example results on SD2 program challenge problems.

Over the course of the SD2 program, we successfully closed the design-build-test-learnloop for engineering organisms with complex circuits. Our BayesDB Whole-genome Simulator augments the output from CELLO, a standard tool for synthetic biologists, to include prediction of the impact of a design on the host. These predictions help synthetic biologists ensure that a proposed design is not toxic. Our approach provides simultaneous predictions for the impact of a design on thousands of host genes, along with continuous measures of host impact as well as interactive visualizations that have proven useful to TA2 Voigt Lab staff. For example, Amin Espah Borujeni (TA2 Voigt Lab)has been working with minimal support to use the BayesDB Whole-genome Simulator hands-on, without our direct involvement, to assist him in making design decisions. We also worked with UCSB to build, design and debug the BayesDB Strain Recommender.This system can be used at the design stage of genetic engineering, allowing designersto choose strains for genetic engineering that are more likely to thrive in vivo when particular genes are knocked out. (Figure 3)

Our successes in SD2 are due to fundamental research in probabilistic programming, an emerging field at the intersection of programming languages and artificial intelligencethat enables the integration of strengths from multiple AI paradigms (Figure 4). The fundamental benefits of probabilistic programming are summarized in Figure 5, and were established via the DARPA PPAML program. We transitioned DARPA PPAML technology to DARPA SD2, enabling us to demonstrate the value of DARPA PPAML results for accelerated scientific discovery and design in data-centric domains. DARPA SD2 also provided funding for basic research and benchmark problems that drove improvements to the performance, expressiveness, and automation of initial results from DARPA PPAML. These basic research improvements have, in turn, transitioned toDARPA SAIL-ON (for AI systems that can adapt to novelty, in some cases by directly applying SD2 open-source systems) and DARPA MCS (for AI systems with more human-like common sense, driven by platform & performance improvements supported by DARPA SD2). See Figure 6 for a summary of the high-level capabilities that have been built over this sequence of DARPA programs and transitions, leading to a scalableprobabilistic programming stack (Figure 6) with broad relevance for the DoD. (Figure 4, Figure 5, Figure 6, and Figure 7).

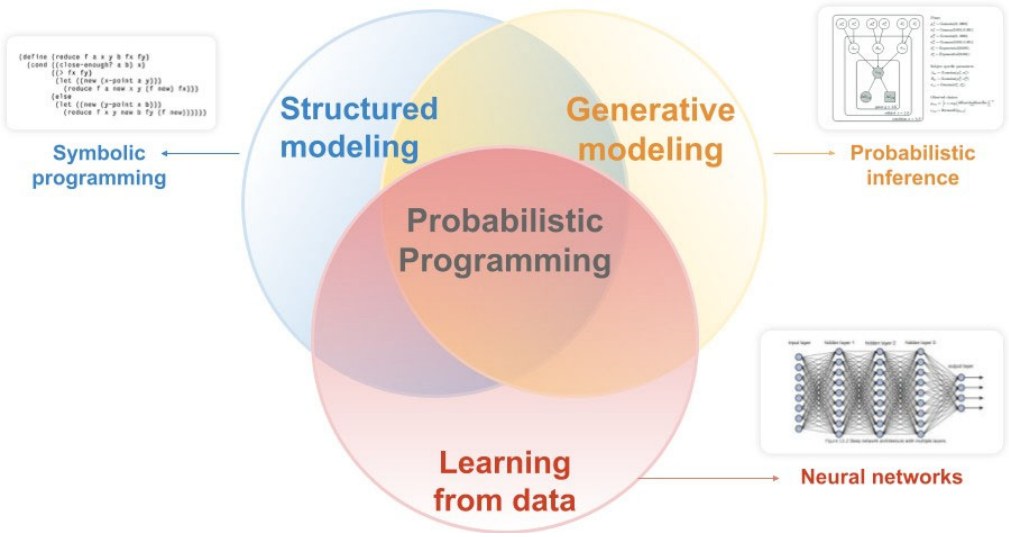


Figure 4. The probabilistic programming framework allows users to combine breakthroughs from different fields and apply them to the hard problem of scientific discovery.

Figure 4. The probabilistic programming framework allows users to combine breakthroughs from different fields and apply them to the hard problem of scientific discovery. A new type of artificial intelligence, probabilistic programming combines structuredknowledge representation and modeling with generative probabilistic modeling and deep learning. This helps users solve a range of problems - from identifying anomalous data in ongoing experiments, to deciding what genes are safe to engineer.

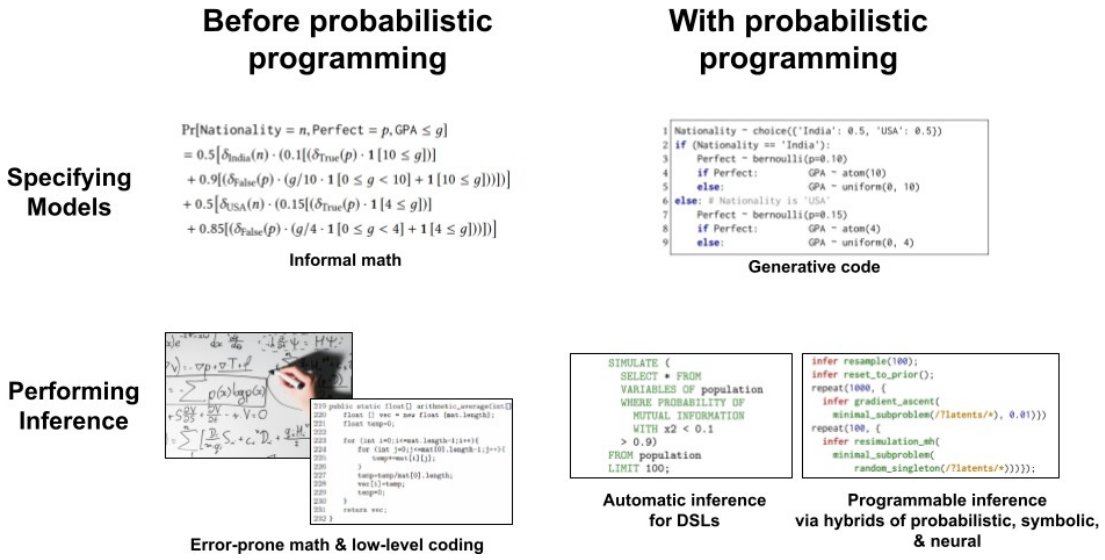


Figure 5. Probabilistic programming yields a factorization of code for modeling and inference, analogous to deep learning platforms such as PyTorch and TensorFlow, but for a much broader class of models and inference algorithms that combine neural, symbolic, and probabilistic approaches.

Figure 5: This in turn increases the scalability, robustness,maintainability, and accessibility of state-of-the-art AI engineering, while simultaneously reducing the development cost.

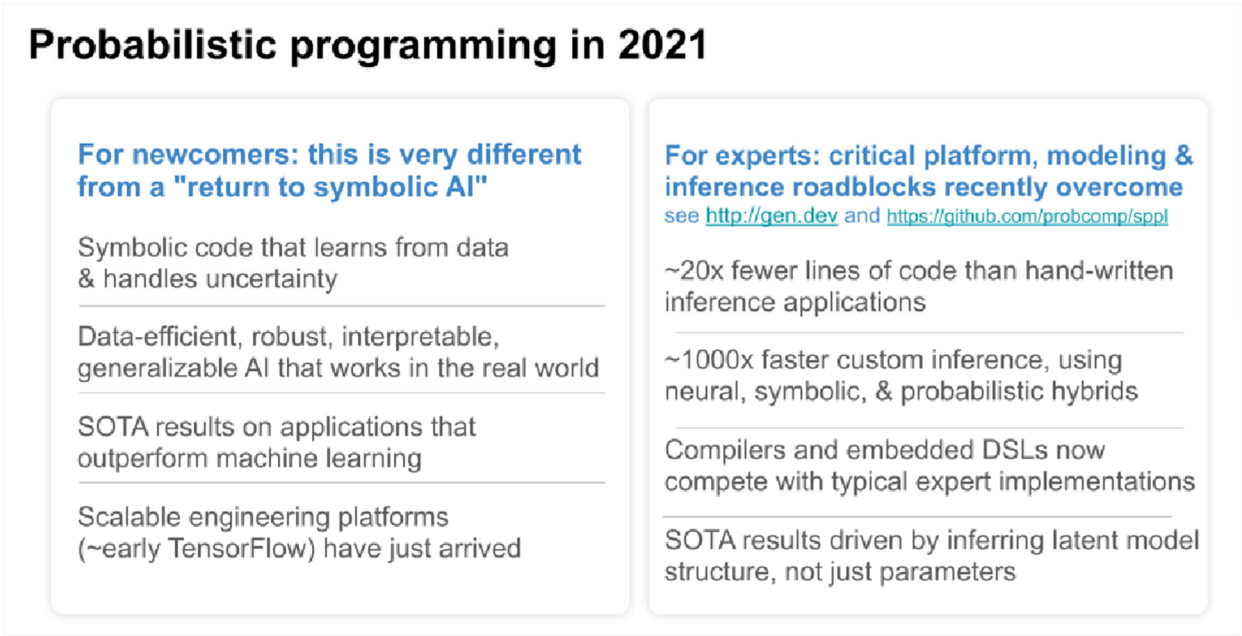


Figure 6. Breakthrough AI capabilities have resulted from DARPA support of probabilistic programming, via platform improvements developed under the PPAML (2013-2017) and SD2 (2017-2021) programs, and capability demonstrations supported under SD2, SAIL-ON

Figure 6. Breakthrough AI capabilities have resulted from DARPA support of probabilistic programming, via platform improvements developed under the PPAML (2013-2017) and SD2 (2017-2021) programs, and capability demonstrations supported under SD2, SAIL-ON (2019-2023), and MCS (2019-2023).

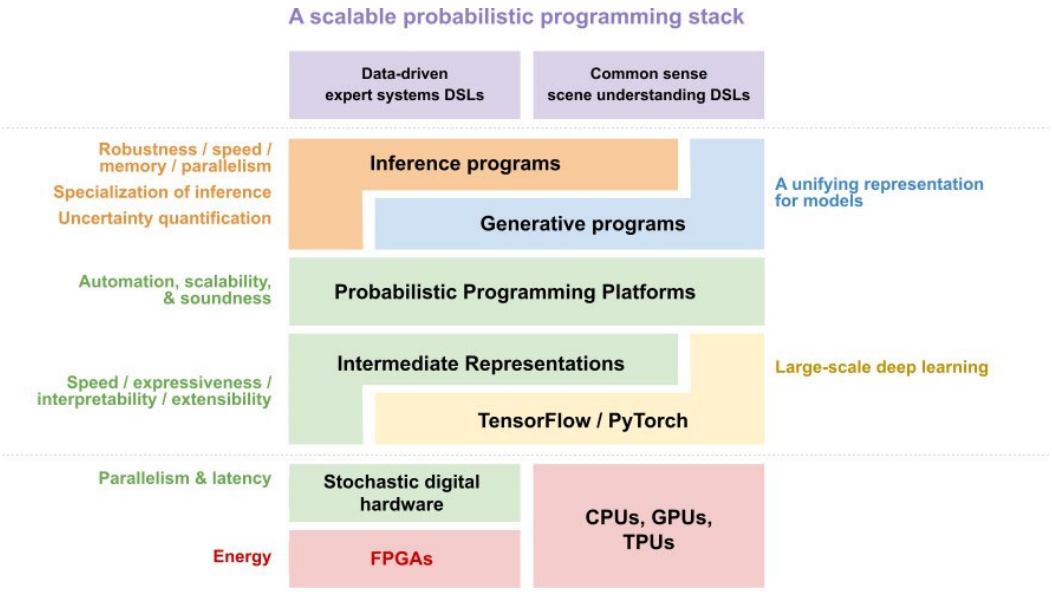


Figure 7. Figure 7. A scalable probabilistic programming stack, spanning AI applications (e.g. data-driven expert systems from DARPA SD2, and common sense scene understanding systems from DARPA MCS) through hardware acceleration for probabilistic inference.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

This section reviews breakthroughs on applications as well as progress on basic research. Applications successes include accelerating design in Novel Chassis by learning whole-genome generative models (3.1), and automatically screening data for anomalies and errors, via the Fail-Fast system (3.2). Basic research successes include learning probabilistic programs for automatic data modeling (3.3.1); fast, exact inferencevia the SPPL query engine (3.3.2); and the development of the Gen probabilistic programming platform (3.3.3).

3.1. Application #1: accelerating design in Novel Chassis

Novel host organisms ("chassis") pose challenging problems for synthetic biologists. The fundamental systems biology that supports design in model organisms such as *E.coli* --- including biochemical pathways and gene regulatory networks --- often is not known. Designs for these organisms tend to be data-driven, requiring extraction of design-relevant information from sparse experimental data, rather than analysis of a large body of background knowledge.

DARPA SD2 developed the Novel Chassis Challenge Problem to drive new data-centrictechniques for discovery and design in precisely these challenging settings. Over the course of the SD2 program, we successfully demonstrated the ability to (i) learn accurate whole-genome simulations for novel organisms from sparse wet lab data; (ii) analyze whole-genome simulations to predict the impact of designs on host organisms, including screening for "safe" genes; and (iii) predict the viability of knock-out strains more accurately than widely-used regression techniques.

3.1.1: Learning whole-genome generative models from high-dimensional wet labdata.

Wet lab experiments in synthetic biology produce "small and wide" data, in which thenumber of variables far outstrips the number of experiments (e.g. ~4000 RNAseq measurements per experiment, but only ~100-1000 experiments).
We created new algorithms for learning whole-genome generative models that simultaneously simulate all ~4000 genes, using explainable models whose structure canbe analyzed and queried in real time. Our BayesDB Whole-genome Simulator predicts novel circuit behavior and how such behavior will impact the rest of a genome.

The BayesDB Whole-genome Simulator shows researchers which genes will interact with one another under any given set of experimental conditions. The models are represented as probabilistic programs and are thus human-readable and explainable. We built the tool in close collaboration with the TA2 Voigt Lab, to ensure that its interface was made with the needs of genetic engineers in mind. (Figure 8) We showed that it is more accurate than widely-used alternative methods, such as linear regression.(Figure 9). The probabilistic programs used inside the BayesDB Whole-genome Simulator are learned from experimental data (Figure 10) via automated Bayesian data modeling techniques that we describe in Section 3.3, and queried via a simple SQL-likelanguage (Figure 11).

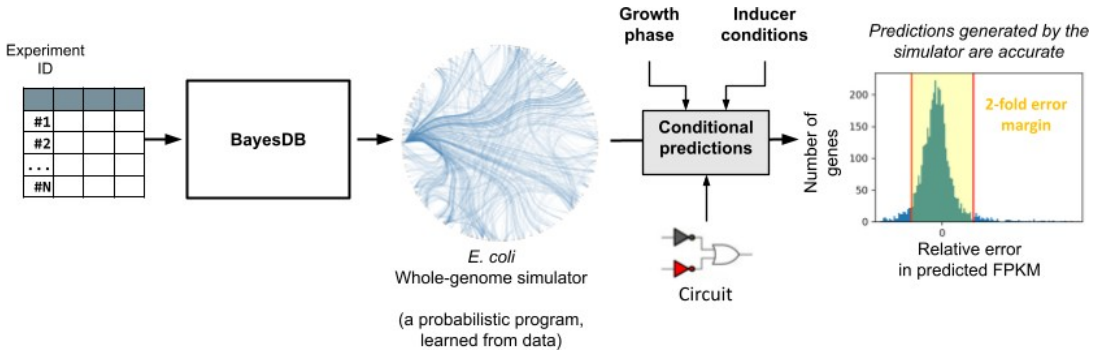


Figure 8. The BayesDB Whole-genome Simulator predicts expression levels of thousands of genes to within 2-fold error (i.e. within expected experimental variability).

Figure 8. The BayesDB Whole-genome Simulator predicts expression levels of thousandsof genes to within 2-fold error (i.e. within expected experimental variability). These predictions can be made for both "wild type" organisms and for engineered organisms under a variety of experimental conditions. A key program accomplishment is the demonstration that learned simulators are accurate to within 2-fold error for over 85% of the ~4000 genes in *E. coli.* This level of accuracy matches expected experimental variability. We have shown that syntheticbiologists find it sufficient to trust the results, and that it works in practice for assisting synthetic biologists in solving design problems.

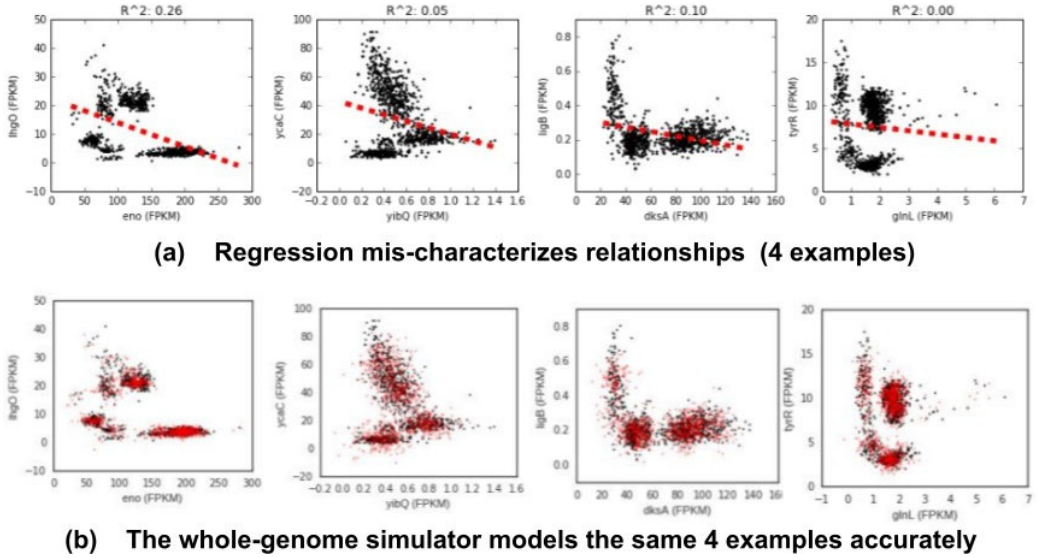


Figure 9. Interactions between the host genes that we detect from NAND2.0 data, but that are missed using linear regression.

Figure 9: (a) shows a closer look at four relationships that were missed by linear regression. The linear fit (dashed, red) to the observed data (black) does not model the dependency adequately. (b) Our whole-genome-simulator, on the other hand, simulates the interactions faithfully to the data. We believe this empirical finding shows a strikingfailure of linear modeling in this domain, that may help to explain synthetic biology design failures, as linear modeling is currently in widespread use.

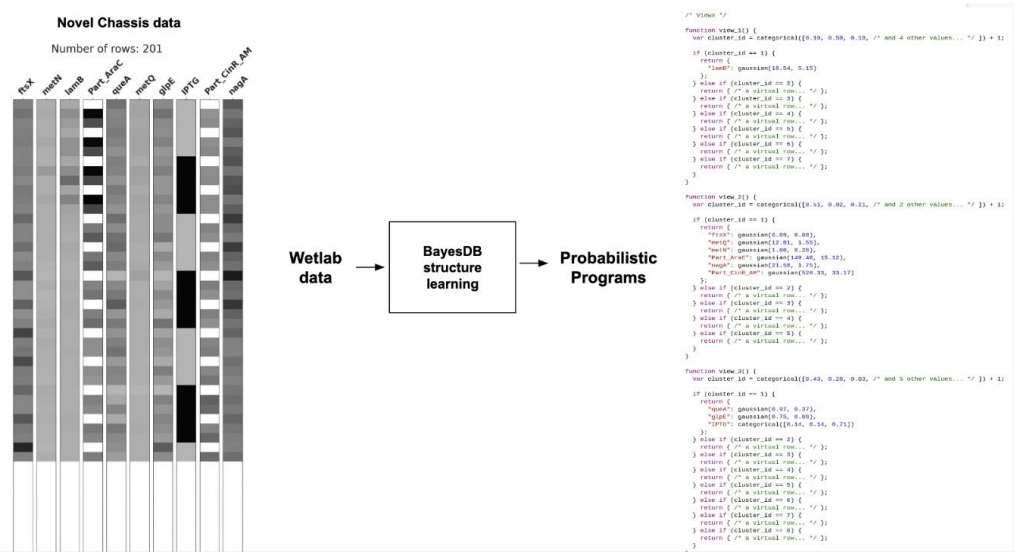


Figure 10. BayesDB learns probabilistic programs from wet lab data.

Figure 10: BayesDB learns probabilistic programs (right) from real wet lab data (left). Each of these probabilistic programs can be viewed as a synthetic data generator that, when run, generates the results from a virtualexperiment. Each program encodes a multivariate model of all the (thousands of) experimental outcomes and variables, approximately characterizing multivariate interactions among them.
BayesDB allows users to easily generate virtual experiments themselves, and also to analyze these probabilistic programs to exactly calculate the probabilities of observed events in real data; to assign rows of real data into operating regimes; and to identify interactions between variables. Also, unlike black-box machine learning models, these probabilistic programs can beedited by humans, to better reflect domain knowledge.

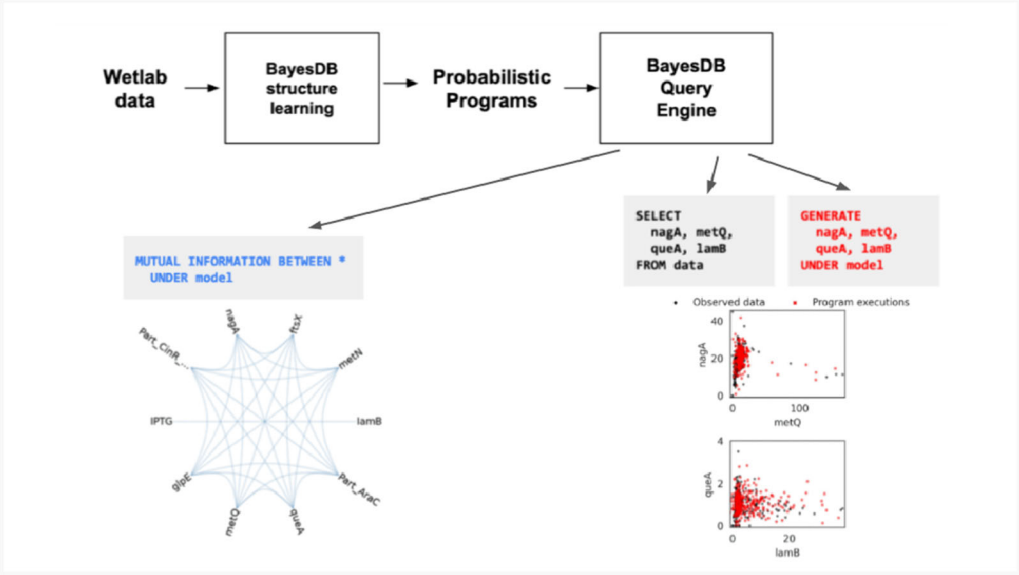


Figure 11. BayesDB enables domain experts to query probabilistic programs using an SQL-like language.

Figure 11: For example, BayesDB users can extract information-theoretic measuresof coupling among variables (bottom left), and compare real data (bottom right, black) with synthetic data (bottom right, red) generated under user-specified constraints.

3.1.2. Screening for "safe" genes.

A key design challenge is to avoid designing toxic organisms, by identifying genes that are "safe" to engineer. Safe designs should minimize impact on host genes that supportbasic survival of the organism.

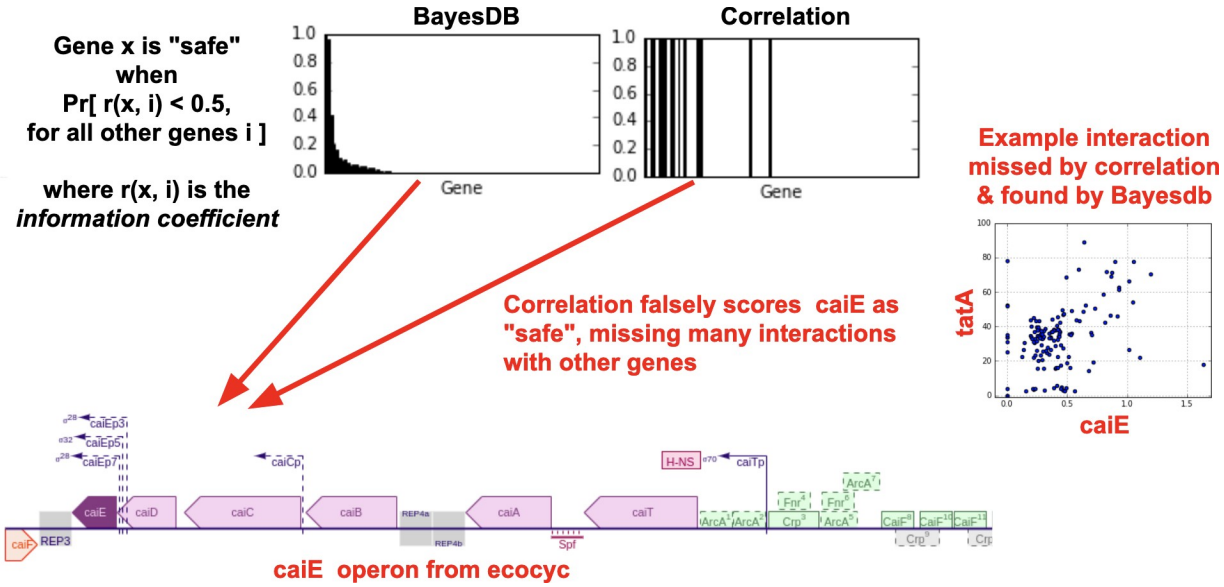


Figure 12. BayesDB screens for genes that are "safe" to engineer.

Figure 12: The *information coefficient r(x, i)* that BayesDB uses to screen genes is derived by analysis of the mutual information among genes from the whole-genome simulator. This analysis detects interactionsbetween genes that are missed by widely-used statistical techniques such as correlation.

We developed an open-source, browser-based application that enables biologists to apply this capability to assist them during design. (Figure 12) No prior knowledge of gene regulatory networks or possible interactions is necessary to use this application. This is particularly helpful for engineers designing in Novel Chassis, because they generally don't know and therefore can't provide information about potential host effects. Once genetic engineers were using our tool to learn about safe sites for genetic engineering by detecting any and all potential interactions, it became clear that a second tool might also be advantageous in the design stage, as opposed to the analysisstage. This led to our development of the BayesDB Strain Recommender (Section 3.1.3).

Figure 13. A screen shot from the BayesDB whole-genome simulator web application, which allows biologists to see the effect a knockout would have without doing a wet lab experiment.

Figure 13: The tool offers three different visualizations of its recommendations, including (i) agiven regime's likelihood (indicated by a blue bar under its name), (ii) which regime it chose forany given simulation (represented by the numbers shown within grey circles), and (iii) plots to show simulations of the whole genome.

3.1.3. Strain recommendation --- predicting strain growth.

The BayesDB Strain Recommender lets designers choose knockout candidates for genetic engineering. For any given gene that synthetic biologists choose to inactivate (e.g. purB in Figure 13), we can assess the effect the knockout has on essential genes in the organism. The strain recommender defaults to considering all genes as potentiallyessential; users can also select the genes on which they want to visualize the effect via a drop-down menu.

The BayesDB Strain Recommender works via whole genome simulation, learning both (i) clusters of genes (regulons) as well as (ii) operating regimes that are specific to thoseclusters. In the example used to generate Figure 13, the user selected six essential genes from a drop-down menu (gatB, rpsS, frr, pheT, fbaA and groES) to see how they

would be effected by a purB knockout. The BayesDB Strain Recommender then stimulates RNA-seq data, and visualizes the three parts of the simulation. First, a givenregime's likelihood is indicated by a blue bar under its name. The darker the blue, the more likely the regulon is to fit into that regime. In this example, setting purB to a low value favors regime_0. Second, the system shows which regime it chose for any given simulation. This is represented by the numbers shown within grey circles, to the right ofthe regime names. The right hand side of the plot shows simulations of the whole-genome.

The BayesDB Strain Recommender allows users to consider partial knock-outs of essential genes. Users who want to see the code underlying these decisions can openthe regime by double-clicking on it, to see the probabilistic JavaScript program that's driving the visualizations and modeling. This is all done via a program that runs within any web browser.

The BayesDB Strain Recommender shows researchers which strains are likely to grow(or not) in a wet lab environment. We worked with UCSB to design, build and test the recommender on commercially available knockouts, using *B. Subtilis* data. UCSB then used those strains to inform their wet lab experiments. We showed the BayesDB StrainRecommender can save time when genetic engineers are deciding which bacterial strains to work with, by identifying which knockout strains have the fewest unwanted effects on the essential genes in an organism *before engineering*. (Figure 14)

The BayesDB Strain Recommender strips away some of the time biologists must spendidentifying strains before their experiments, and debugging on the other end when experiments fail. We hope the tool will be made available to even more researchers via designnovelchassis.org, to help them successfully select strains for the badly understood process of genetic engineering. The tool is already available via a passwordprotected webpage for SD2 participants.

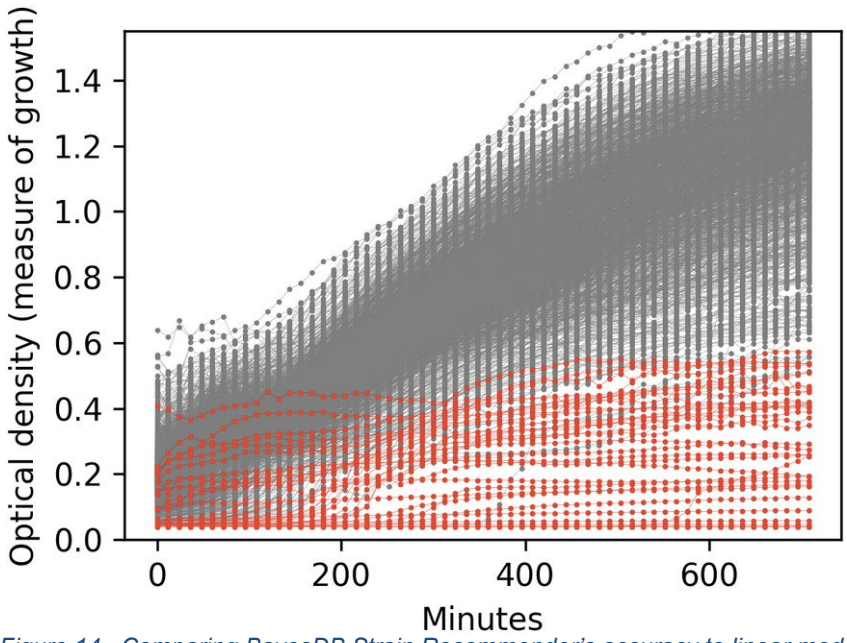


Figure 14. Comparing BayesDB Strain Recommender's accuracy to linear modeling.

Figure 14: The BayesDB Strain Recommender correctly flags all 36 "bad" strains (shown in red), while linearregression flagged only 31 of these strains, falsely labeling 5 "bad" strains as likely to thrive.

3.2. Application #2: automatically screening data for errors

A widespread problem in wet lab work is that it is often impossible to detect critical data& protocol errors before the final stages of data analysis. This leads to wasted time andmoney, and slows the pace of discovery and design. A key goal of SD2 was to developdata-driven technology that respected domain knowledge and leveraged expected coherence of results across multiple labs, to detect errors and anomalies early in the process, and therefore reduce the time and cost of discovery and design.

We created SD2 Fail-Fast, an open-source system that automatically screens multivariate data from complex real-world experiments to detect probable data errors, protocol violations, and anomalous results. SD2 Fail-Fast detects deeper semantic errors than simple deterministic rules, by automatically learning explainable, symbolic probabilistic programs that model multivariate data streams (leveraging domain knowledge) and using them to infer probable errors, protocol violations, and anomalousvalues.

3.2.1. Fail-Fast system overview.

The Fail-Fast system allows users to rapidly screen data for errors using automatically learned multivariate probabilistic programs. Users can specify meta-data describing real-world experiments, including the types of variables and high-level "invariants" (e.g.expected qualitative associations between variables). Fail-Fast then flags data for

unlikely cells and violations of invariants. These capabilities are implemented by BayesDB queries. In SD2, the Fail-Fast system (Figure 15) was used to solve problemsin multiple domains, as is described in 3.2.2 and 3.2.3 below.

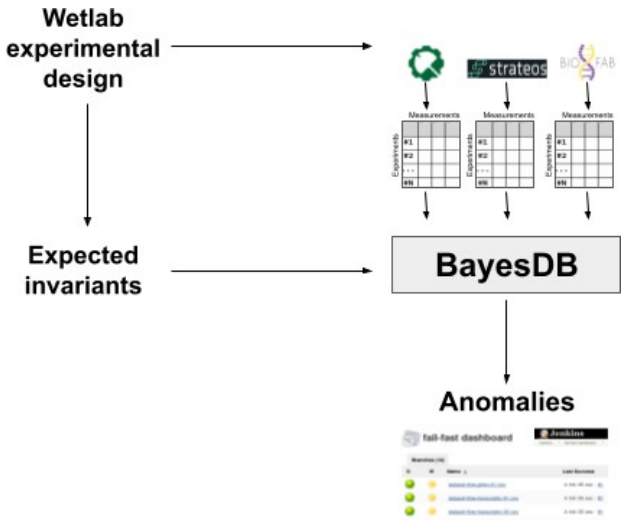


Figure 15. The BayesDB Fail-Fast pipeline in SD2.

Figure 15: The tool was able to confirm (i) the detectionof false dependencies on ProStab, during year 2, where one data source was highly predictive of stability but should not have been, and (ii) protocol violations relating to arabinose induction at 18 hours in Novel Chassis that occurred in year 3. Additional information is provided in Sections 3.2.2 and 3.2.4.

3.2.2. ProStab application.

SD2 Fail-Fast detected data errors in SD2 ProStab and other early SD2 challenge problems in experimental data stemming from labs in CA, MA and WA. In 2018, our team was able to detect an anomaly in the initial ProStab data release in only two hours, using BayesDB (Figure 16 left). After working with us to identify the root of the flaw, UW re-issued the dataset, potentially saving multiple performers many days of wasted effort. BayesDB also found there were large differences between the ProStabtraining data and testing data. The drift between testing and training data led to BayesDB reporting it was highly uncertain about the stability of almost all proteins. In this case, BayesDB's uncertainty can be attributed to a lack of signal for predicting stability in the test data. Indeed, scatter plotting data from the different predictors showed that there was a drift between training and test data (see Figure 16 right). Wereported this result and our concern that the root issue was data drift to the data creators, who incorporated our feedback into the next round of wet lab data.

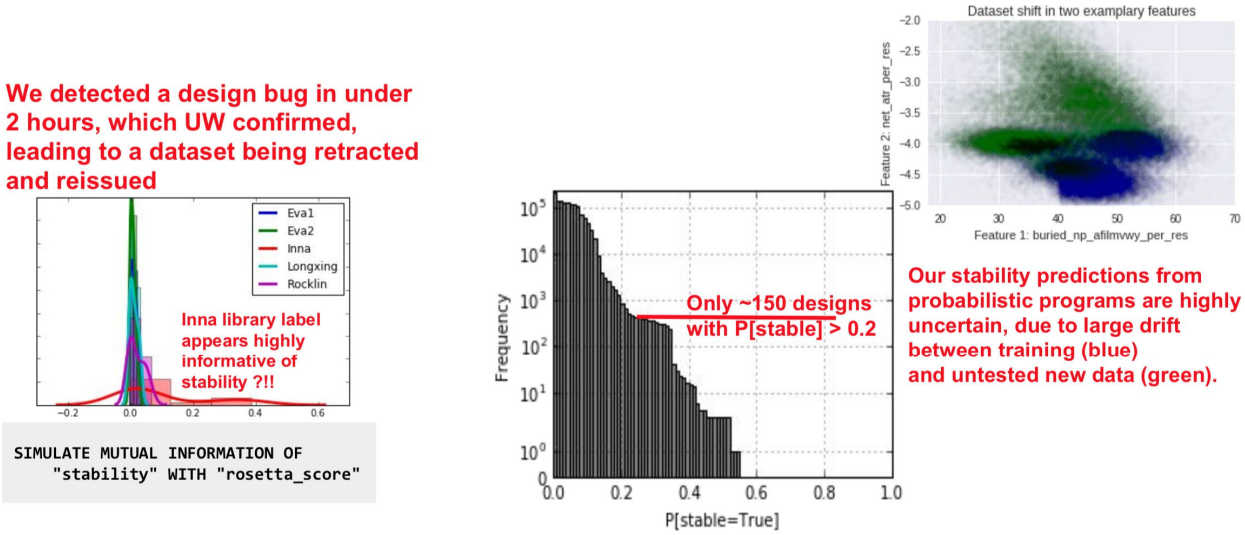


Figure 16. Fail-Fast detects problems in protein stability data.

Figure 16: On the left, we show a highmutual information statistic between one of the ProStab data sources (Inna, red) and protein stability. The middle plot depicts BayesDB remaining uncertain about Protein stability of test data. This is because BayesDB detected dataset drift between training and test data (right).

3.3.1. Yeast States application.

The Fail-Fast system was tested with data from the Yeast States challenge problem. The goal was to identify genetic circuits that behave in unexpected ways, based on flowcytometry data; see Figure 17 for an example. Early in the SD2 project, this goal was one of the key drivers behind SD2 work on innovation representing experimental intent (e.g. should the circuit output be 1 or 0?) and representation of scientific knowledge (e.g. which strain implements which circuit diagram?).

This application tested a knowledge-driven variant of the Fail-Fast system, in which

(i) we wrote custom probabilistic programs to encode correct circuit behavior, and

(ii) their parameters were tuned via inference algorithms, to better match empirical data.

Custom probabilistic programs for this application were implemented using the Ventureprogramming language [3, 4] from DARPA PPAML. These probabilistic programs explicitly modeled the circuit and provided mechanistic models of circuit failure. This knowledge-intensive approach, based on manually authored probabilistic programs (including custom inference programs; see Figure 18) is complementary to a more data-driven approach that relies on probabilistic programs learned via Bayesian structure learning techniques.

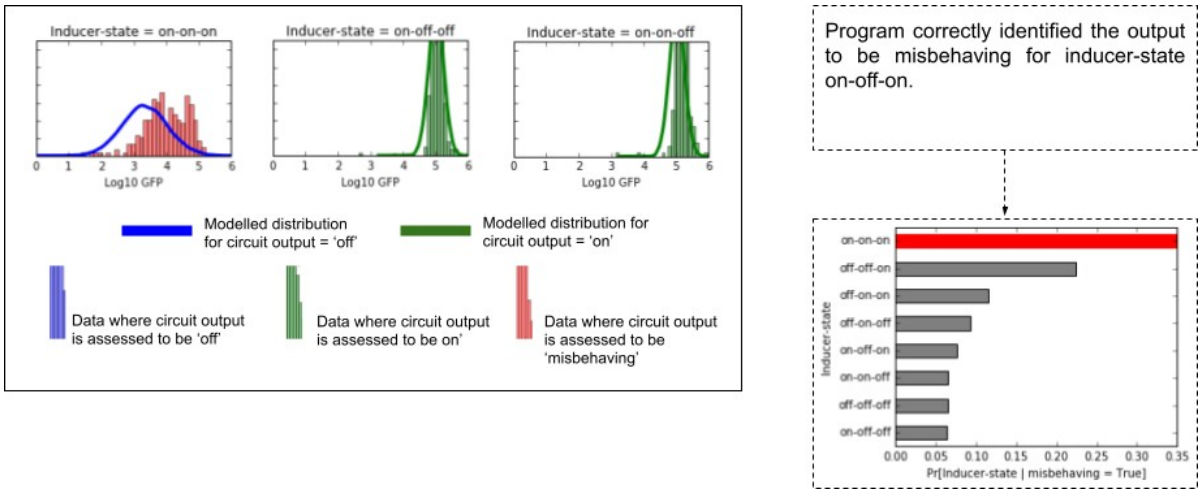


Figure 17. Fail-Fast can detect errors in circuit behavior.

Figure 17: This application relies on customprobabilistic programs encoding expected circuit outcomes given inducer states, going beyondthe automatically learned models used for results on ProStab in Figure 16.

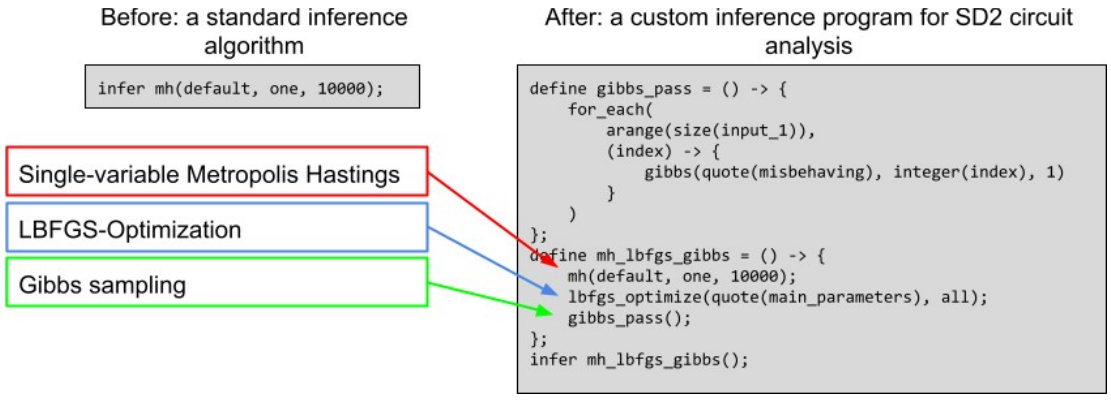


Figure 18. Custom inference algorithms used to adjust parameters of mechanistic failure models on Yeast States data.

Figure 18: These custom inference programs enabled us to fit parametersof probabilistic models to data, and to accurately infer circuit failure modes.

3.3.2. Arabinose depletion.

We also found our Fail-Fast system could solve unexplained problems that were first surfaced by the BayesDB Whole-genome Simulator. While making predictions simultaneously for ~4000-5000 gene expression levels, the BayesDB Whole-genome Simulator flags anomalies using statistics of the distribution on residuals from these predictions. The high-dimensional predictive models that enable this kind of predictive anomaly detection are difficult to construct via standard methods from machine learning and computational statistics.

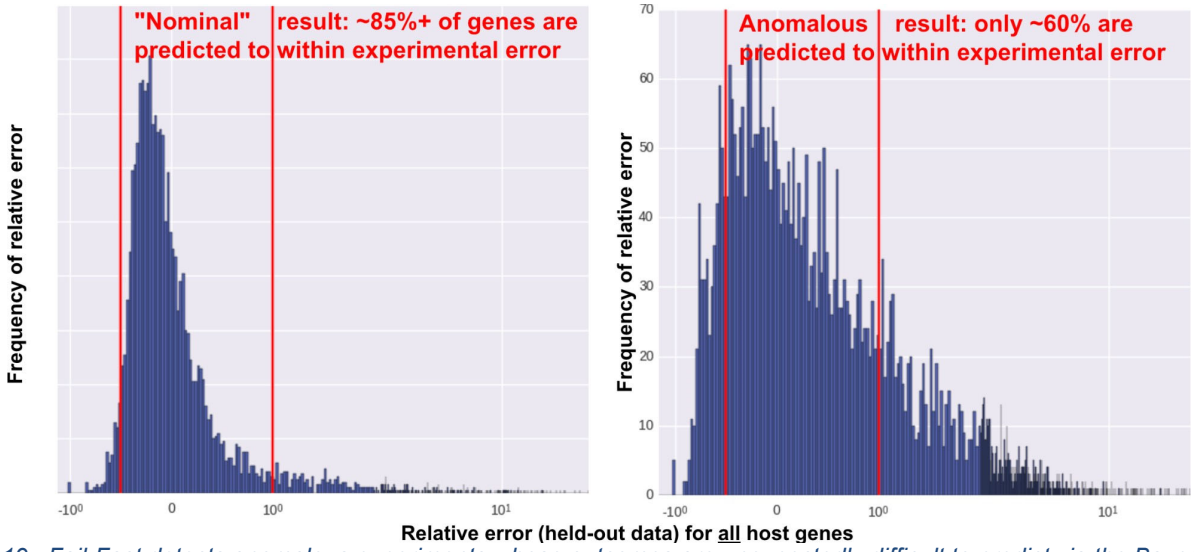


Figure 19. Fail-Fast detects anomalous experiments whose outcomes are unexpectedly difficult to predict via the BayesDB Whole-genome Simulator.

Figure 19: Fail-Fast detects anomalous experiments whose outcomes are unexpectedly difficult to predict via the BayesDB Whole-genome Simulator. (left) Typical experiments allow for ~85%+ of genes to be predicted to be within the range of experimental error. (right) An example experimental anomaly, in which only ~60% of genes can be predicted within the rangeof experimental error via the BayesDB Whole-genome Simulator. This decrease in predictabilityreflected an anomaly that was confirmed by the experiment's designers. This is an example of the Fail-Fast system's ability to automatically detect data errors.

Using a dataset from the Novel Chassis challenge problem, our BayesDB tool found justsuch a data error in predictions for a NAND circuit induced with Arabinose, as is shown in Figure 19. Adding the Arabinose reagent provides input to the NAND gate. The figureshows two different experimental conditions during this induction: (a) growth phase of the organism and (b) the organism's stationary phase. We see a much larger error in the predictions for (b). This was unexpected. We presented the error during the Q1 2020 quarterly meeting as an anomaly that our tool had flagged, but could not explain.
Following the presentation, the experiment's designers reached out to us confirming thatwe found a data problem.

The problem was that testing at 18 hours was too late, as the bacteria had consumed the Arabinose during the stationary phase, before the measurement was taken. This means that while the experimental variables implied that the NAND circuit was inducedwith Arabinose, it effectively wasn't, and any Arabinose-induced reactions in the organism could not happen. This manifested in a misalignment between experimental protocol and experimental reality-- as the recorded measurements were de facto showing an organism not induced with Arabinose.

It is exciting to see that our probabilistic programming approach can automatically detect these kinds of errors, without requiring hard-coding of a domain theory. Structurelearning is able to produce models from wet lab data that are sufficiently flexible and accurate to reveal these kinds of errors.

3.3.3. Transition to Takeda for clinical trials.

Our BayesDB technology and Fail-Fast use cases were successfully transitioned to support a two-year sponsored research project with Takeda Pharmaceuticals, to test ourability to automatically screen experimental data from clinical trials for data entry/ETL errors, anomalies, and potential violations of protocol inclusion/exclusion criteria (Figure20). DARPA SD2 supported the open-source software development and basic researchthat enabled this project. MIT staff collaborated across both SD2 and Takeda, with Takeda responsible for funding the Takeda-specific clinical trials application (and contributing supplemental funding for open-source and some basic EECS/PPL research).

Takeda employees are using the Fail-Fast approach to monitor clinical trials, which often include complex combinations of hundreds of different types of variables, includinghealth outcomes, subject self-reports and clinician's assessments of symptoms, as well as basic background demographics such as BMI and age group. (Figure 21) Clinical trial monitoring is challenging not only because of the many types of data, but also because in most cases, a majority of data values are missing, with more data missing toward the end of the trial. Despite these challenges, Fail-Fast can identify data quality problems and violations of analysis plans and eligibility criteria at all stages of a trial. (Figure 22)

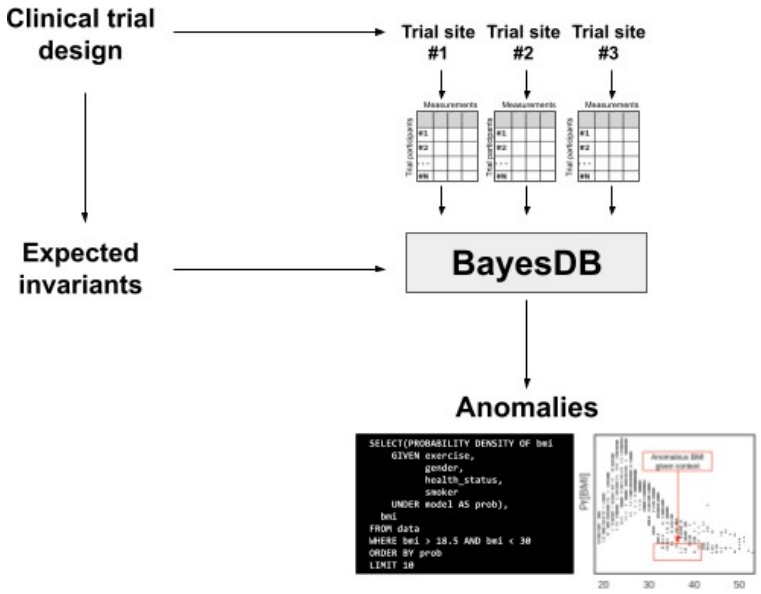


Figure 20. BayesDB Fail-Fast transition to clinical trial data, sponsored by Takeda Pharmaceuticals.

Figure 20: Takeda has been funding its own project using Fail-Fast in the clinical trial quality assurance pipeline to identify site performance issues, identify data entry errors, and detect violations of inclusion or exclusion criteria.

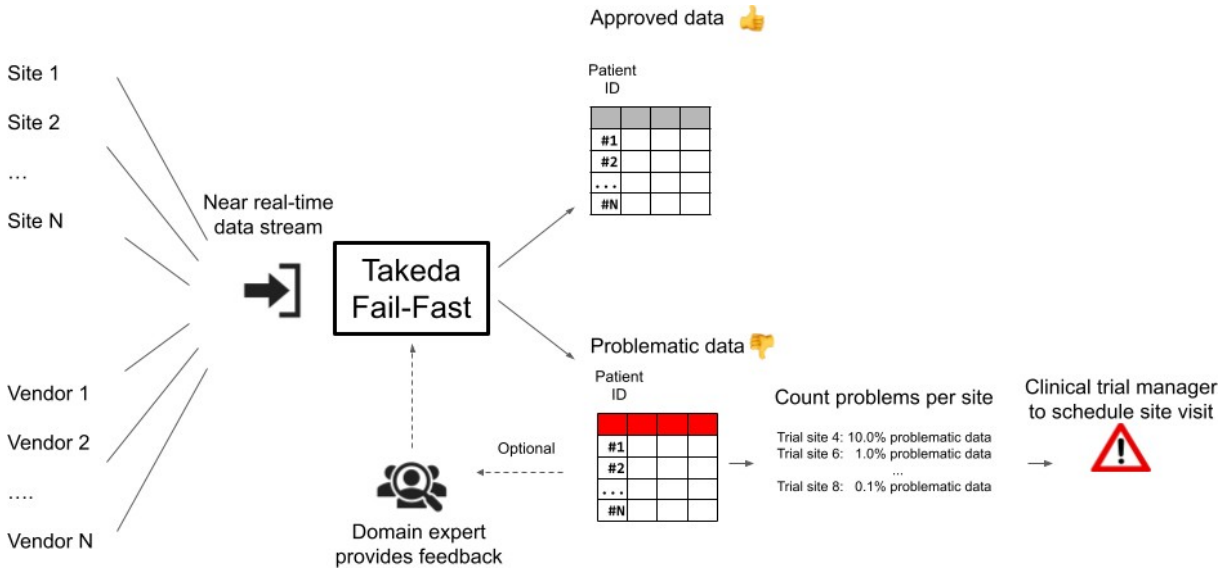


Figure 21. An overview of Takeda's Fail-Fast use case.

Figure 21: Trial data from many sites and vendors is input into the system, which then screens the data for probable anomalies. Somedata gets approved. Other data is found to be problematic. The problematic data is automatically flagged by the tool so that it can be manually reviewed by Takeda staff.

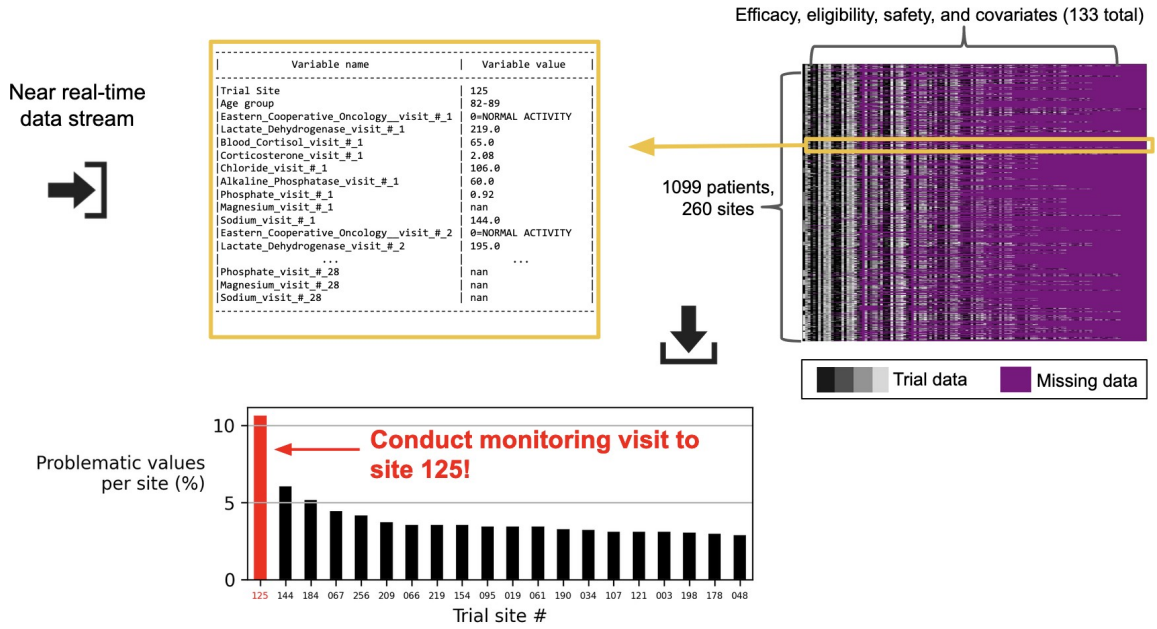


Figure 22. The Fail-Fast approach applied to Takeda's data measured hundreds of types of variables, modeled the data, and identified sites with problematic data values.

Figure 22. Our funded work with Takeda showed that even though data streams from real-world clinical trials are complex, Fail-Fast can model the data (and its dependencies) to identify potential anomalies. In this case, the tool flagged the site with the highest fraction of problematic values. The tool found that data stemming from site 125 had roughly double the amount of problems asother sites. The high fraction of problematic data values for site 125 might warrant an intervention: the tool therefore flagged the site for review by human observers, who would then choose whether to schedule a site visit.

3.4. Basic research on probabilistic programming

SD2 drove several basic research advances in probabilistic programming that enabled and tested the applications capabilities described in Sections 3.1 and 3.2, and that alsocontributed more broadly to the emerging field of probabilistic programming.

3.4.1. Learning Probabilistic Programs for Automatic Data Modeling.

SD2 targeted data-driven, knowledge-poor domains. These domains differ from standard applications of probabilistic programming in Bayesian statistics and someapplications in cognitive AI. In contrast to SD2, standard probabilistic programmingapplications have used probabilistic programs primarily to encode richer prior knowledge than is feasible via ML techniques.

SD2 requires data-driven learning of probabilistic programs from sparse data. This is afundamental AI and machine learning problem of broad importance. Our research yielded new solutions to this problem. These solutions are not only applicable to "smalland wide" multivariate SD2 data from wet lab experiments, but also to univariate and multivariate time series (Figure 23) and to relational databases.

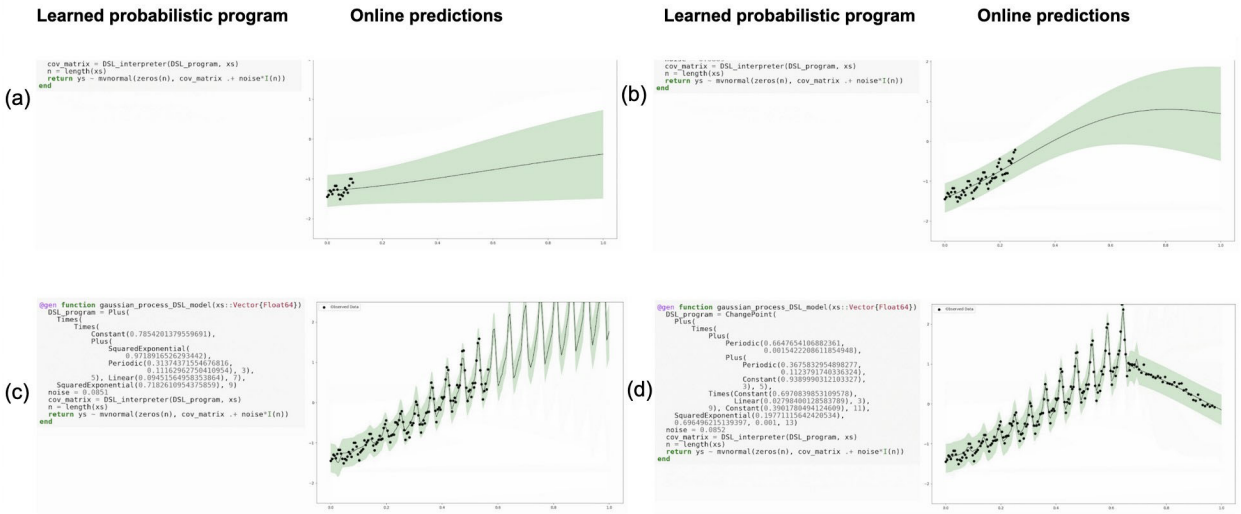


Figure 23. An example of online structure learning for probabilistic programs.

Figure 23: Code for a learned generative program is on the left side of each graph. The predictions from that generative program are shown on the right. The program learns, from a-d: it starts by explainingthe data with noise and a linear trend (a) and (b), then shifts toward periodic structure and increasing amplitude with confident extrapolations (c), and then when a changepoint comes in, the system rapidly detects and adapts its model to that change (d).

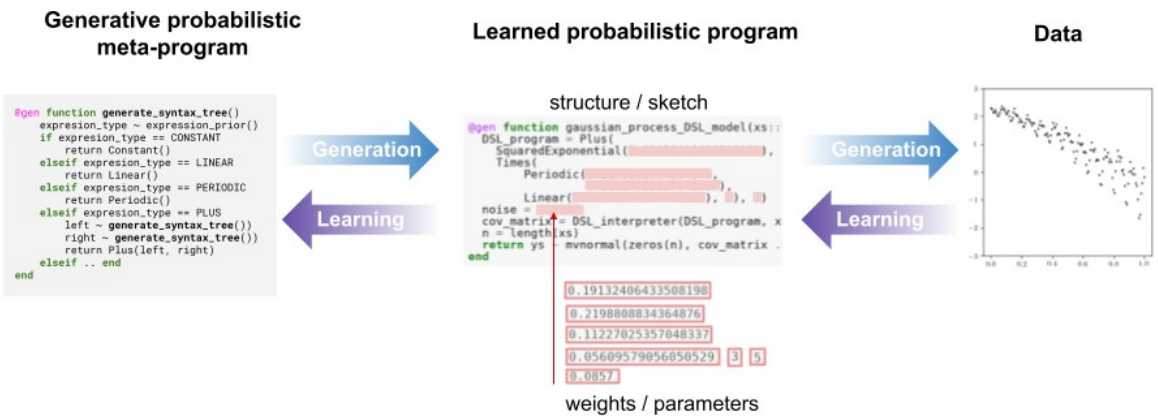


Figure 24. The technical approach used for learning the structure of probabilistic programs.

Figure 24: This approach relies on generative meta-programs that generate probabilistic programs, that in turn generate observable data. Probabilistic inference is then used to learn probabilistic programs that are compatible with both the generative meta-program (encoding theprior) and the empirical data. Crucially, the generative meta-program prior encodes a broad state of ignorance over possible probabilistic programs that could have generated the data. Thisuse of probabilistic programming to encode broad ignorance priors is important for robust learning from sparse data.

Learning probabilistic programs from sparse data required fundamental algorithmic advances and served as a challenging test for probabilistic programming infrastructure originally developed under DARPA PPAML [5,6]. Consider that when learning the structure of probabilistic programs, the structure of the target program changes in the inner loop of inference. This means that to perform gradient optimization over program parameters, we need to operate over a latent computation graph whose structure is being inferred and mutated over the course of inference. In contrast, deep learning algorithms only need to calculate gradients for a single network structure that is fixed during the course of parameter estimation. Changing the structure of the programs is itself a challenging problem (Figure 24). Structure changing moves need to compare probabilistic programs with varying numbers of parameters, whose densities lie in spaces with different dimensions. We've invented generalizations of well-known reversible-jump MCMC algorithms --- and automated them using combinations of automatic differentiation and probabilistic programming --- to handle the demands of thissetting. (Figure 25)

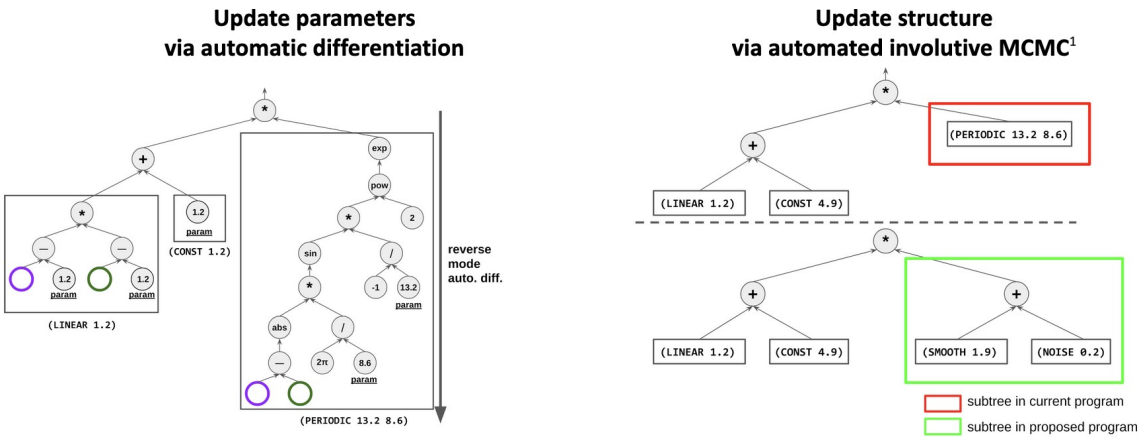


Figure 25. Inference challenges in structure learning that we solved during the course of the SD2 program.

Figure 25: The left panel shows an example computation graph representing a single probabilistic program arising during the course of structure learning. This computation graph supports automatic differentiation, to enable fast gradient-based updates to program parameters, interleaved with structure changing moves. The right panel shows an example change in program structure, replacing the periodic kernel on top (which expands into the computation graph on the right branch of the top-level tree in the left panel) with a sum of a smoothness kernel and noise kernel, shown on the bottom. These structure changes are implemented via automated involutive MCMC techniques that we developed during SD2.

3.4.2. Speeding up our tools with fast, exact inference via the SPPL query engine.

When we began, probabilistic programs were widely believed to be too slow and inaccurate for many practical applications. (Consider that our own early whole-genomesimulators took hours to screen for safe genes.) The key computational bottleneck wasinference: taking a probabilistic program and using it to either generate simulations for or calculating probabilities of events of interest.

We sped up inference ~1000x during SD2, for a broad class of probabilistic programs that are sufficient for many data-driven discovery and design applications. Specifically, we developed the Sum-Product Probabilistic Language (SPPL, Figure 26, Figure 27), anew probabilistic programming system that delivers exact probabilities and simulations, with ~1000x faster results than the previous state-of-the-art [7]. SPPL makes it practical to calculate probabilities of rare events in milliseconds, and use probabilistic inference inreal-time interfaces, e.g. by biologists using SPPL programs to screen designs for impact on host organisms.

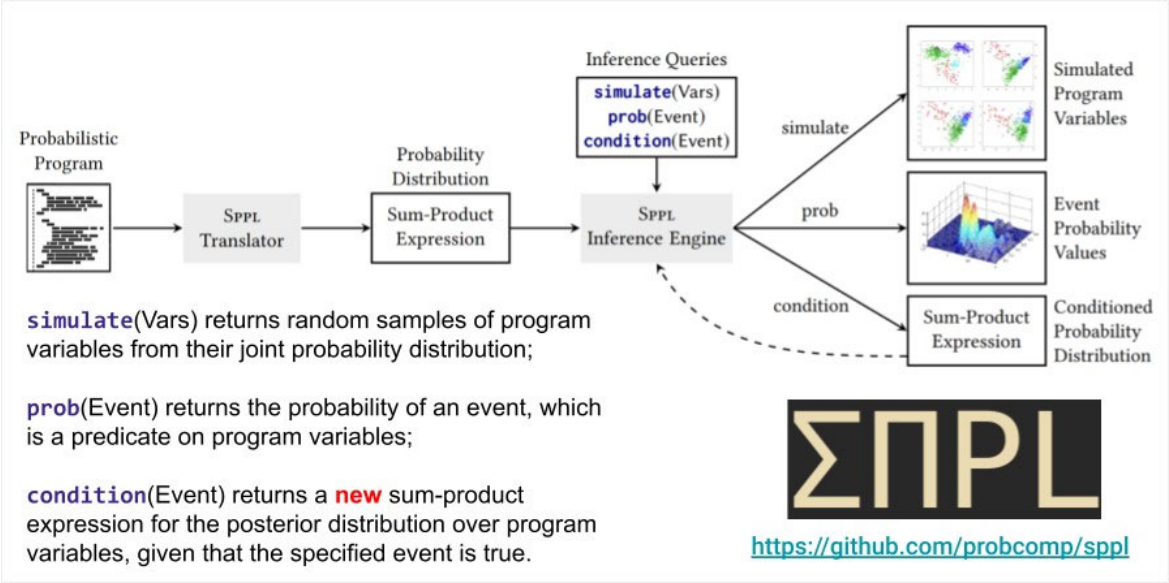


Figure 26. The SPPL language provides fast, exact, symbolic inference via translation of probabilistic programs into sum-product expressions.

Figure 26: Sum-product expressions, describedin [7], allow for provably correct Bayesian inference in high-dimensional probabilistic programs containing both discrete and continuous variables. These capabilities significantly expand the domain of applications for which probabilistic programming is a practical solution, and proved critical for our applications research during SD2.

The SPPL programs we used in SD2 were learned from data via the techniques from Section 3.3.1. Our technical approach thus involves composing two basic research successes:

1. Learning the structure of SPPL programs from experimental data
2. Using fast exact symbolic inference for SPPL programs to produce exactinference results that answer questions posed by synthetic biologists

It is the combination of these two innovations that enabled us to apply probabilistic programming to data-driven domains with thousands of variables and deliver answers that helped synthetic biologists with discovery and design. Bayesian structure learning could produce accurate models and account for uncertainty, even given the large number of variables and lack of prior knowledge. Exact symbolic inference allowed us toquery the models and confidently produce exact results that synthetic biologists could rely on, without having to worry about scalability of inference or approximation error (unlike with previous probabilistic programming systems; runtime comparisons can be seen in Figures 28 and 29, and Table 1).

We believe this combination of structure learning and exact symbolic inference could be transformative for many other data-driven domains.

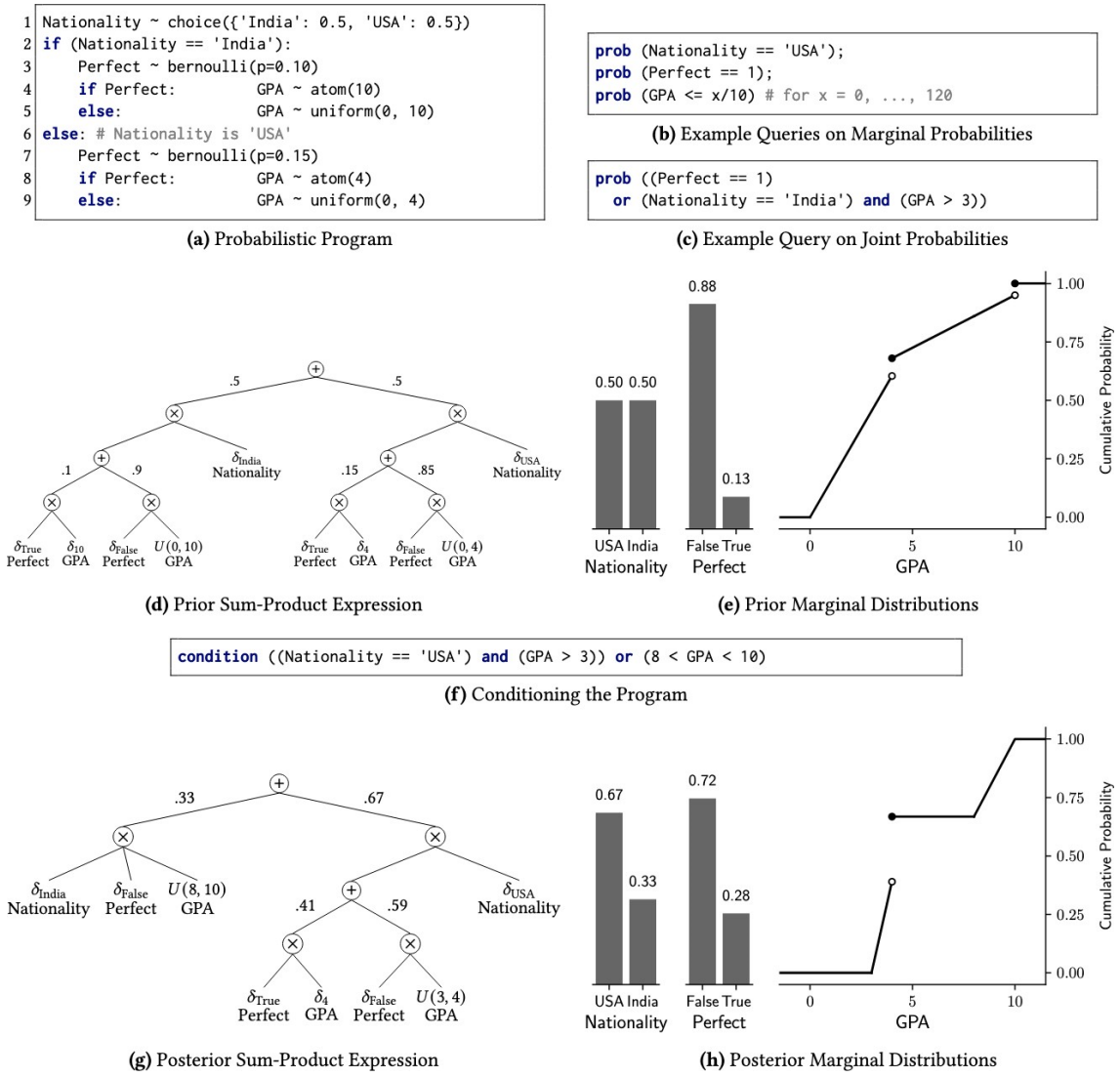


Figure 27. An example of exact inference using sum-product expressions in SPPL.

Figure 27: This example, taken from the paper “SPPL: Probabilistic Programming with Fast Exact Symbolic Inference” [7], shows SPPL programs and sum-product expressions for both the prior distribution (a, d) and the posterior distribution (g) given a compound constraint (f) in the "IndianGPA problem", an example probabilistic program with both continuous and discrete random variables. SPPL results include exact probability densities (b, c) and posterior marginal distributions (h) obtained via exact symbolic inference.

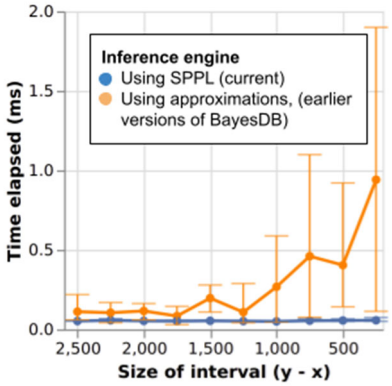


Figure 28. BayesDB with accelerated SPE inference is faster and more predictable than earlier versions of the prototype.

Figure 28: BayesDB with SPPL (shown in Blue), which uses sum product expressions to return exact results, is faster than earlier versions of BayesDB, which used a CrossCat inference engine (shown in orange) and returned only approximate results. The earlier version became slower as the queries users input became more complicated -- theruntime remains constant in the new version.

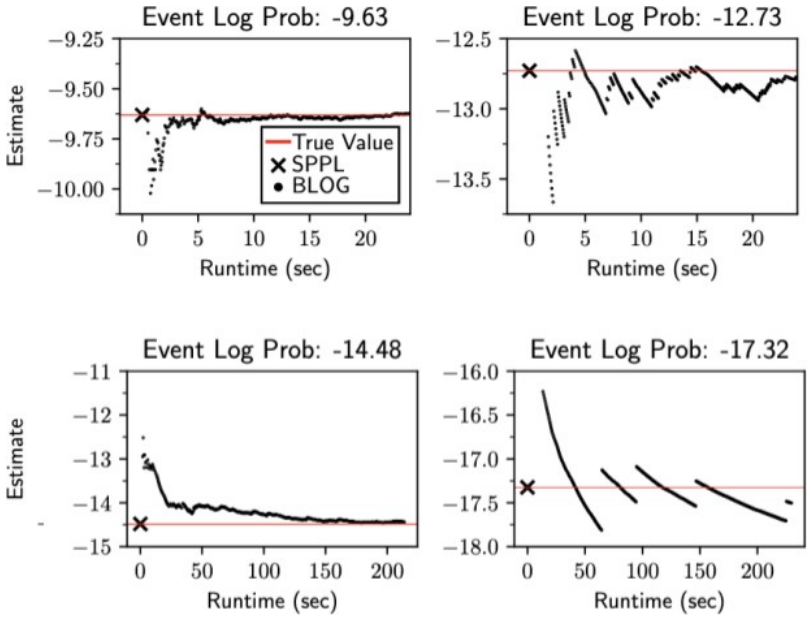


Figure 29. SPPL is faster than other programs that can do the same thing.

Figure 29: This shows a runtime comparison for computing probabilities using exact inference in SPPL and rejectionsampling in the probabilistic programming language BLOG.

Table 1. SPPL is faster than other benchmarks systems.

Decision Program	Population Model	Lines of Code	Fairness Judgment	Wall-Clock Runtime (seconds)			SPPL Speedup Factor	
				FairSquare	VeriFair	SPPL	vs. FairSquare	vs. VeriFair
DT_4	Independent	15	Unfair	1.4	16.0	0.01	**140x**	**1600x**
	Bayes Net. 1	25	Unfair	2.5	1.27	0.03	**83x**	**42x**
	Bayes Net. 2	29	Unfair	6.2	0.91	0.03	**206x**	**30x**
DT_{14}	Independent	32	Fair	2.7	105	0.03	**90x**	**3500x**
	Bayes Net. 1	46	Fair	15.5	152	0.07	**221x**	**2171x**
	Bayes Net. 2	50	Fair	70.1	151	0.08	**876x**	**1887x**
DT_{16}	Independent	36	Fair	4.1	13.6	0.03	**136x**	**453x**
	Bayes Net. 1	49	Unfair	12.3	1.58	0.08	**153x**	**19x**
	Bayes Net. 2	53	Unfair	30.3	2.02	0.08	**378x**	**25x**
DT^{α}_{16}	Independent	62	Fair	5.1	2.01	0.06	**85x**	**33x**
	Bayes Net. 1	58	Fair	15.4	21.6	0.12	**128x**	**180x**
	Bayes Net. 2	45	Fair	53.8	24.5	0.12	**448x**	**204x**
DT_{44}	Independent	93	Fair	15.6	23.1	0.05	**312x**	**462x**
	Bayes Net. 1	109	Unfair	264.1	19.8	0.09	**2934x**	**220x**
	Bayes Net. 2	113	Unfair	t/o	20.1	0.09	–	**223x**

Runtime measurements and speedup factors on a benchmark set off 15 fairness verification tasks using SPPL, FairSquare[8], and VeriFair [9]. For more information on how SPPL outperforms previous systems for verifying decision trees, see the SPPL paper [7].

3.4.3. Development of the Gen probabilistic programming platform.

When we began, probabilistic programs were only applied in narrow domains such as Bayesian statistics and data analysis. Inference was not customizable enough or fast enough for many real-world applications. This was a fundamental challenge surfaced bythe DARPA PPAML program.

To address this, we created Gen, a general-purpose probabilistic programming platformthat makes it practical to apply probabilistic inference to solve hard problems in systemsbiology and computer vision, among other domains (Figures 30, 32, 33, [10-13]). The metrics show that Gen programs (i) can require ~20x fewer lines of code than typical expert implementations but with performance that is competitive with typical handwrittenimplementations by experts; (ii) support hybrids of neural, symbolic, and Monte Carlo inference that deliver SOTA accuracy; and (iii) are ~1000x faster than previous probabilistic programming languages with custom inference. (Figure 31, [10])

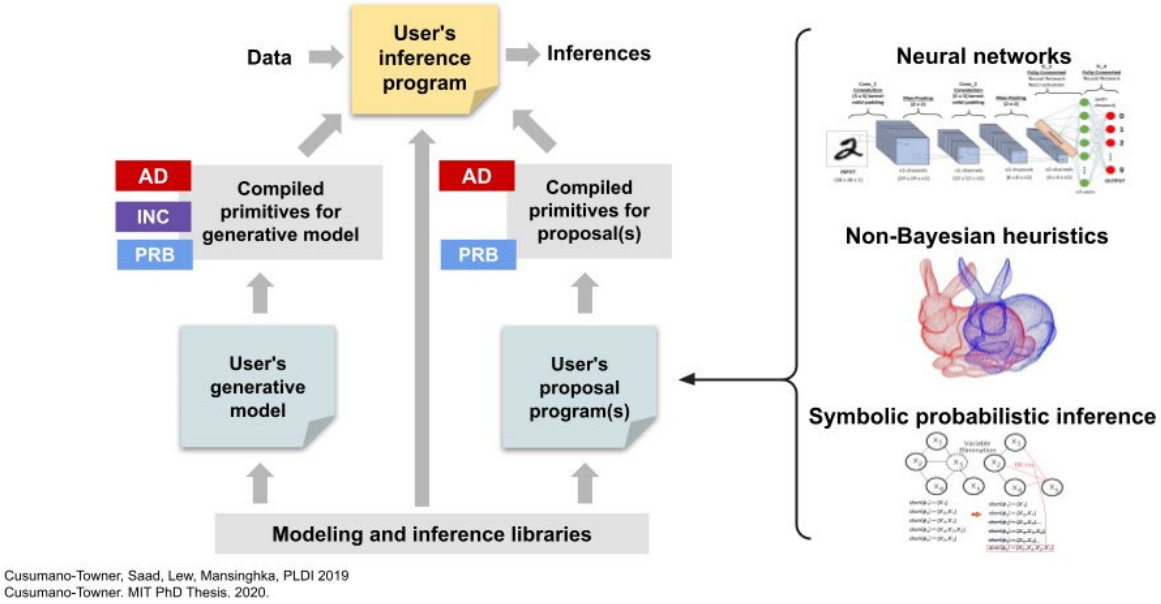


Figure 30. Architecture of a typical Gen inference application.

Figure 30: Gen supports multiple inference paradigms, including dynamic programming, variational inference, MAP optimization,sequential Monte Carlo and Markov Chain Monte Carlo. In this figure, AD refers to automatic differentiation; INC to incremental computation; and PRB to probabilistic sampling and scoring.These three operations (AD, INC, and PRB) are the fundamental building blocks needed to implement a broad range of inference, optimization, and learning algorithms.

~20x fewer lines of code

Gen	Automated Statistician[1] (MATLAB + Python)
130 / 260 lines of code	4,166 lines of code

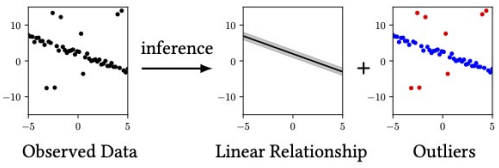
Improved performance

	Caching	Runtime (ms/step)
Gen (DML)	Provided by Recurse	2.57ms (± 0.09)
Julia (Handcoded)	None	4.73ms (± 0.45)
Gen (DML)	None	6.21ms (± 0.94)
Venture	None	279ms (± 31)

Figure 31. Gen can implement structure learning of probabilistic programs using ~20x fewer lines of code than previous approaches, while delivering improved performance.

Figure 31: These benchmark results, from [10], show how Gen performs on structure learning problems.Gen is the first probabilistic programming system that makes it practical to experiment with a broad class of structure learning approaches.

(a) Probabilistic inference task



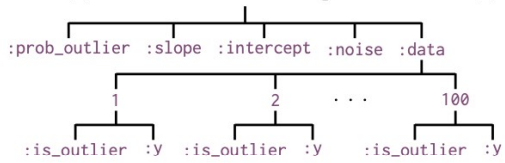
(b) Generative model

```
@gen function generate_datum(x, prob_outlier, noise, (grad)(slope),
        (grad)(intercept))::Float64
  if @trace(bernoulli(prob_outlier), :is_outlier)
    (mu, std) = (0., 10.)
  else
    (mu, std) = (x * slope + intercept, noise)
  end
  return @trace(normal(mu, std), :y)
end

generate_data = Map(generate_datum)

@gen (static) function model(xs)
  slope = @trace(normal(0, 2) :slope)
  intercept = @trace(normal(0, 2) :intercept)
  noise = @trace(gamma(1, 1) :noise)
  prob_outlier = @trace(uniform(0, 1) :prob_outlier)
  n = length(xs)
  ys = @trace(generate_data(xs, fill(prob_outlier, n),
      fill(noise, n), fill(slope, n), fill(intercept,n)), :data)
  return ys
end
```

(c) The hierarchical address space of `model` in (b).



(d) Helper Julia function for making observations

```
function make_observations(ys::Vector{Float64})
  observations = choicemap()
  for i=1:length(ys)
    observations[:data => i => :y] = ys[i]
  end
  return observations
end
```

(e) Custom proposal used by inference programs (f) and (g)

```
@gen function is_outlier_proposal(previous_trace, i::Int)
  is_outlier = previous_trace[:data => i => :is_outlier]
  p_outlier = is_outlier ? 0.0 : 1.0
  @trace(bernoulli(p_outlier), :data => i => :is_outlier)
end
```

(f) User inference program 1

```
function inference_program_1(xs::Vector{Float64}, ys::Vector{Float64})
  # constrain observed random variables to their observed values
  constraints = make_observations(ys)

  # obtain an execution trace of model that satisfies the constraints
  (trace, _) = generate(model, (xs,), constraints)

  # iteratively update the values of latent variables
  for iter=1:100
    # update four parameters all at once
    selection = select(:prob_outlier, :noise, :slope, :intercept)
    (trace, _) = metropolis_hastings(trace, selection)

    # update outlier classifications one by one, using a custom proposal
    for i=1:length(xs)
      (trace, _) = metropolis_hastings(trace, is_outlier_proposal, (i,))
    end
  end

  return trace
end
```

(g) User inference program 2

```
function inference_program_2(xs::Vector{Float64}, ys::Vector{Float64})
  # constrain observed random variables to their observed values
  constraints = make_observations(ys)

  # initialize slope and intercept using least squares regression
  (slope, intercept) = least_squares(xs, ys)
  constraints[:slope] = slope
  constraints[:intercept] = intercept

  # obtain an execution trace of model that satisfies the constraints
  (trace, _) = generate(model, (xs,), constraints)

  # iteratively update the values of latent variables
  for iter=1:100
    # update custom subsets of parameters, repeating several times
    for j=1:5
      (trace, _) = metropolis_hastings(trace, select(:prob_outlier))
      (trace, _) = metropolis_hastings(trace, select(:noise))
      (trace, _) = mala(trace, select(:slope, :intercept), 0.001)
    end

    # update outlier classifications one by one, using a custom proposal
    for i=1:length(xs)
      (trace, _) = metropolis_hastings(trace, is_outlier_proposal, (i,))
    end
  end

  return trace
end
```

(h) Runtime versus accuracy of inference programs in (f) and (g).

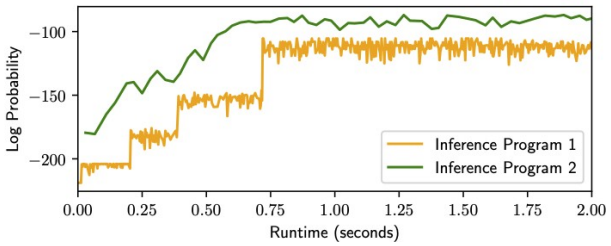


Figure 32. Example of modeling and inference programming in Gen, from [10].

Figure 32: Consider a robust regression task (a). Users define generative models (b) and custom proposal distributions (e) in embedded modeling languages. Users also implement inference programs (f)–(g) in the host language (Julia), using methods provided by the inference library (shown in bold). (h) shows the time-accuracy profiles of the two inference programs, which implement inference algorithms that have different efficiencycharacteristics.

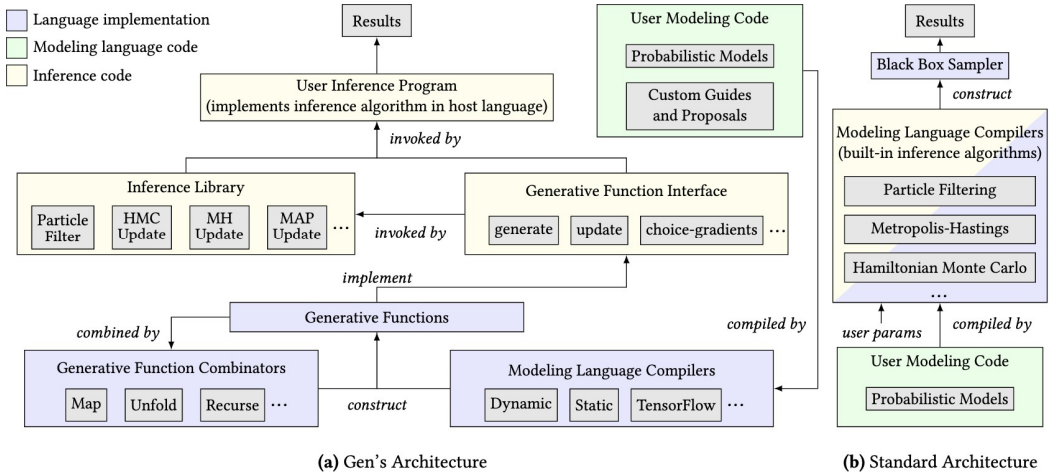


Figure 33. Comparison of Gen's architecture (a, left) to a standard probabilistic programming architecture (b, right), from [10].

Figure 33: Gen allows users to specify custom models and inference algorithmsin high-level languages, unlike standard PPLs, which require users to use a small set of black-box inference algorithms.

Gen has been used by many other researchers over the course of the SD2 program. For example, it has been used to learn causal models from observational data [14], for probabilistic cleaning of tabular data [15], for research on metacognition [16], and for network-based epidemiological simulations [2]. It's become clear since the open sourcerelease of Gen that it meets a need of researchers across domains who are not connected with MIT. Users at UW Madison have independently applied Gen to learn the causal network structures of signaling pathways from phosphorylation time series data to better understand cell regulation, opening up new approaches to fundamental data-driven modeling challenges that are relevant for discovery and design [1]. In their paper, published in *Bioinformatics,* they explained why they chose Gen for their experiments (Figure 34):

2.3.2 Implementation in Gen

We chose the Gen PPL (Cusumano-Towner *et al.*, 2019) for its expressive power and customizable inference. While implementing SSPS, the customizability of Gen allowed us to begin with simple prototypes and then makes successive improvements. For example, our model initially used a dense *adjacency matrix* representation for G, but subsequent optimizations led us to use a sparse *parent set* representation instead. Similarly, our MCMC method started with a naïve 'add or remove edge' proposal distribution; we arrived at our sparse proposal distribution (Section 2.2) after multiple refinements. Other PPLs do not allow this level of control.

Figure 34. Comparison of Gen's architecture (a, left) to a standard probabilistic programming architecture (b, right), from [10].

Our Intel, IBM, and DARPA MCS transition partners value Gen's unique ability to deliverreal-time inference of uncertain structure from real-world data --- not just estimate parameters --- and to enable new hybrids of neural, symbolic, and probabilistic AI techniques. Intel, IBM, and DARPA MCS have provided follow-on funding, and are using Gen as the basis for fundamental new research in building AI with human-like common-sense. Large-scale research tests are underway via the DARPA MCS program, though industry/defense development (via a sustained team of ~5 engineers) is needed for operational use of Gen in production AI engineering.

4.0 RESULTS AND DISCUSSION

In our proposal, we wrote that if we were maximally successful, the SD2 program could catalyze the creation of a collaborative knowledge-production ecosystem in which AI software could collaborate with human researchers to produce knowledge that is immediately useful for solving discovery & design problems in data-centric domains.[17]

Our SD2 results show that this vision is scientifically achievable in the domain of synthetic biology, using research breakthroughs in probabilistic programming initially funded by DARPA. We also began transitioning our open-source prototypes to other usecases, both within DARPA and funded by industry.

The strongest evidence for success are the results from the BayesDB Whole-Genome Simulator and BayesDB Strain Recommender. Both of these systems generate modelsthat combine qualitative knowledge, quantitative modeling assumptions, and empirical data to create recommendations for genetic engineering. The tools allow users to run “virtual wet labs” that generate data from hypothetical RNA experiments for any organism, without having any prior knowledge about the organism. This allows users toquickly, automatically identify suitable knockout strains “in silico” to improve genetic engineering success rates in wet labs, and quantify unwanted effects on the host organism caused by genetic circuit parts -- without doing any real-world engineering.
This information can be used by a tool like Cello to inform design. The automatically learned models allow biologists without computer science backgrounds to apply probabilistic programming to predict the results of future experiments. The tools are alsointeractive; scientists can query the models to guide experimentation and robust design.

Improvements to our infrastructure were continuous over the course of the program. Webriefly summarize some quantitative highlights of these results (Figure 35).

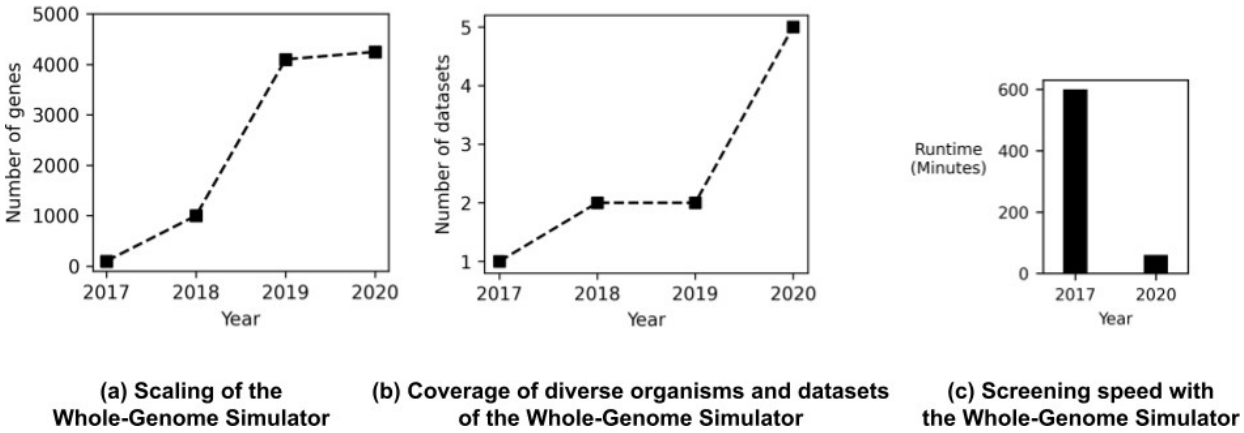


Figure 35. Metrics quantifying improvements during SD2.

Figure 35: (a) Our initial prototype could onlybuild simulators for 100 genes**.** As scalability improved, we were able to analyze 1000 genes (but ran into limitations in our query engine). By 2019, we could handle ~4000+ genes, addressing the needs of our biologist collaborators. (b) As automation improved, and as interfaces for checking model quality became easier to use, we were able to broaden coverage to build simulators for multiple datasets from more organisms (*Y. Cerevisae, E. coli, E. coli Nissle, B. subtilis, Pseudomonas putida).* (c) Introducing SPPL reduced the time needed for analyzing all pairs of genes in the host to ~1 hour, from ~10 hours via our initial prototype.

5.0 CONCLUSIONS

The DARPA Synergistic Discovery and Design (SD2) program set out to “developdata-driven methods to accelerate scientific discovery and robust design in domains that lack complete models.” Our MIT research team has successfully demonstrated data-driven solutions to multiple hard discovery & design problems in synthetic biology, leveraging breakthrough probabilistic programming research results developed over thecourse of the SD2 program. Specific problems we have addressed include (i) screeningwet lab data for errors / anomalies; (ii) screening for genes that are "safe" to engineer;
(iii) predicting host impact of potential designs; and (iv) predicting growth rates for strains. (v) We have also developed open-source prototypes implementing some of these research results that have collectively attracted over $2M of transition funding,from both industry (via Takeda Pharmaceuticals, IBM, and Intel) and DARPA (via theMCS and SAIL-ON programs).

The central scientific advance enabling our success is new AI technology that learns grounded, data-driven probabilistic programs from real-world wet lab experiments. These probabilistic programs can generate synthetic values for thousandsof measurements and tens of experimental and design variables, screen new experiments for errors, and classify experimental samples into empirically-discovered "operating regimes". In some cases, they can act like a "virtual wet lab", filling an analogous role to classical computational modeling techniques, but apply to data-drivendomains where even qualitative causal knowledge is not available. These probabilistic programs can also be analyzed for causal information about the biological system understudy. Furthermore, many of these capabilities are accessible via a simple, SQL-like language that is intended to be learnable by biologists (and experts in other domains)who lack an understanding of data science.

This level of AI assistance for discovery and design could potentially have long-term, transformative impact across multiple STEM fields. Our SD2 results primarilydemonstrated efficacy for synthetic biology, and our transitions substantiate value for medical research. Clinical trials and clinical research are an especially important application area; reducing the data and time needed for clinical discoveries could save lives and reduce unnecessary human suffering. We also believe that, longer term, SD2's core capabilities around data-centric discovery could open up new opportunities for the DoD and federal government in policy and in operational decision support. AI assisted model discovery and decision support from sparse administrative and operational data could add value by improving the quality of data, by surfacing relationships that might not be apparent to human experts, and by providing automated,data-driven decision support.

6.0 REFERENCES

[1] Merrell, David, and Anthony Gitter. 2020. “Inferring signaling pathways with probabilistic programming.” *Bioinformatics* Volume 36, no. Issue Supplement_2 (December): Pagesi822–i830.

[2] Smedmark-Margulies, Niklas; Walters, Robin; Zimmermann, Heiko; Laird, Lucsas; van der Loo, Christian; Kaushik, Neela; Caceres, Rajmonda; van de Meent, Jan-Willem. 2021 “Probabilistic Program Inference in Network-based Epidemiological Simulations”. Posterpresented at non-archival *PROBPROG*, October.

[3] Mansinghka, V.K.; Selsam, D.; and Perov, Y. 2014 “Venture: A higher-order probabilistic programming platform with programmable inference.*arXiv preprint*, arXiv:1404.0099.

[4] Mansinghka, V.K.; Schechtle, U.; Handa, S.; Radul, A.; Chen, Y.; and Rinard, M. 2018 “Probabilistic programming with programmable inference.” In *PLDI 2018: Proceedings ofthe 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 603–616. ACM

[5] Schaechtle, Ulrich, Saad, Feras, Alexy Radul, and Vikash K. Mansinghka. 2017. “Time seriesstructure discovery via probabilistic program synthesis.” arXiv:1611.07051. 2017.

[6] Saad, F. A.; Cusumano-Towner, M. F.; Schaechtle, U.; Rinard, M. C.; and Mansinghka, V. K. “Bayesian synthesis of probabilistic programs for automatic data modeling.” *Proceedingsof the ACM on Programming Languages, 3(POPL)*: 37:1-37:29. January 2019.

[7] Saad, Feras A., Martin C. Rinard, and Vikash K. Mansinghka. 2021. “SPPL: Probabilistic Programming with Fast Exact Symbolic Inference.” *PLDI: Proceedings of the 42nd ACMSIGPLAN International Conference on Programming Language Design and Implementation*, (June), 804-819.

[8] Albarghouthi, Aws, Loris D’Antoni, Samuel Drews, and Aditya V. Nori. 2017. “FairSquare: Probabilistic Verification of Program Fairness.” *Proc. ACM Program. Lang. 1, OOPSLA,Article 80*.

[9] Bastani, Osbert, Xin Zhang, and Armando Solar-Lezama. 2019. “Probabilistic Verification ofFairness Properties via Concentration.” *Proc. ACM Program. Lang. 3, OOPSLA, Article118*, (October).

[10] Cusumano-Towwner, Marco F., Alex K. Lew, and Vikash K. Mansinghka. 2019. “Gen: a general-purpose probabilistic programming system with programmable inference.” *PLDI2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 221-236. https://dl.acm.org/doi/10.1145/3314221.3314642.

[11] Cusumano-Towner, Marco F. 2020. “Gen: A high-level programming platform for probabilistic inference,” *PhD Thesis.* Massachusetts Institute of Technology, Cambridge,Massachusetts.

[12] Cusumano-Towner, Marco F., and Vikash K. Mansinghka. 2018. “A design proposal for Gen: Probabilistic programming with fast custom inference via code generation.”

MAPL2018: Workshop on Machine Learning and Programming Languages (co-located with PLDI 2018), 52-57.

[13] Cusumano-Towner, Marco F., Feras A. Saad, Alex K. Lew, and Vikash K. Mansinghka. 2018. *"Gen: A general-purpose probabilistic programming system with programmableinference Technical Report MIT-CSAIL-TR-2018-020,"* Technical Report

MIT-CSAIL-TR-2018-020, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology,. Computer Science and Artificial IntelligenceLaboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[14] Witty, Sam, Kenta Takatsu, David Jensen, and Vikash Mansinghka. 2020. "Causal Inference using Gaussian Processes with Structured Latent Confounders." *ICML 2020:Proceedings of the Thirty-Seventh International Conference on Machine Learning, ofProceedings of Machine Learning Research*.

[15] Lew, A. K.; Agrawal, M.; Sontag, D.; and Mansinghka, V. K. 2021. "PClean: Bayesian Data Cleaning at Scale with Domain-Specific Probabilistic Programming." In *AISTATS 2021: Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*,volume 130, of *Proceedings of Machine Learning Research*, pages 1927–1935, 2021. PMLR

[16] Berke, Marlene, Mario Belledone, and Julian Jara-Ettinger. 2020. "Learning a metacognitionfor object perception." *SVRHM workshop at NeurIPS*.

[17] Mansinghka, Vikash. 2017. *BAA - HR001117S0003 Technical Area 1 Proposal: BayesDBfor Data Centric Scientific Discovery*.

7.0 PUBLICATIONS & PRESENTATIONS (alphabetical by author by year)

Saad, F.; and Mansinghka, V. “Detecting dependencies in sparse, multivariate databases usingprobabilistic programming and non-parametric Bayes.” In *AISTATS 2017: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, of *Proceedingsof Machine Learning Research*, pages 632–641, Fort Lauderdale, Florida, 2017. PMLR

Saad, F.; Casarsa, L.; and Mansinghka, V. K. “Probabilistic search for structured data via probabilistic programming and nonparametric Bayes.” *arXiv preprint*, arXiv:1704.01087. 2017.

Schaechtle, U.; Saad, F.; Radul, A.; and Mansinghka, V. K. “Time series structure discovery viaprobabilistic program synthesis.” *arXiv preprint*, arXiv:1611.07051. 2017.

Cusumano-Towner, M. F.; Bichsel, B.; Gehr, T.; Vechev, M.; and Mansinghka, V. K. “Incrementalinference for probabilistic programs.” In *PLDI 2018: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 571–585, 2018.
ACM

Cusumano-Towner, M. F.; and Mansinghka, V. K. “A design proposal for Gen: Probabilistic programming with fast custom inference via code generation” In *MAPL 2018: Workshop on Machine Learning and Programming Languages* (co-located with PLDI 2018), pages 52–57,2018.

Cusumano-Towner, M. and Mansinghka, V. “Using probabilistic programs as proposals.” *arXivpreprint:* 1801.03612. Publication date 01/2018.

Cusumano-Towner, M. F.; Saad, F. A.; Lew, A.; and Mansinghka, V. K. *Gen: A general-purposeprobabilistic programming system with programmable inference.* Technical Report MIT-CSAIL-TR-2018-020, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, November 2018

Mansinghka, V. K.; Schaechtle, U.; Handa, S.; Radul, A.; Chen, Y.; and Rinard, M. Probabilistic programming with programmable inference. In *PLDI 2018: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 603–616,2018. ACM

Saad, F.; and Mansinghka, V. K. “Temporally-Reweighted Chinese Restaurant Process Mixturesfor Clustering, Imputing, and Forecasting Multivariate Time Series.” In *Artificial Intelligence and Statistics 21 (AISTATS)*, volume 84, pages 755–764. PMLR, 2018.

Schaechtle, Ulrich. “Automated data modeling for science via Bayesian probabilistic programsynthesis.” Invited talk at *PROBPROG* 2018, October 2018.

Bolton, A. D.; Haesemeyer, M.; Jordi, J.; Schaechtle, U.; Saad, F. A.; Mansinghka, V. K.; Tenenbaum, J. B.; and Engert, F. “Elements of a stochastic 3D prediction engine in larvalzebrafish prey capture.” *eLife*, 8: e51975. November 2019.

Cusumano-Towner, M. F.; Saad, F. A.; Lew, A. K.; and Mansinghka, V. K. “Gen: A general-purpose probabilistic programming system with programmable inference.” In

Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019), Phoenix, AZ, USA, 2019.

Saad, F. A.; Cusumano-Towner, M. F.; Schaechtle, U.; Rinard, M. C.; and Mansinghka, V. K. “Bayesian synthesis of probabilistic programs for automatic data modeling.” *Proceedings of theACM on Programming Languages, 3(POPL)*: 37:1-37:29. January 2019.

Saad, F. A.; Freer, C. E.; Ackerman, N. L.; and Mansinghka, V. K. “A family of exact goodness-of-fit tests for high-dimensional discrete distributions.” In *AISTATS 2019: Proceedingsof the 22nd International Conference on Artificial Intelligence and Statistics*, volume 89, of *Proceedings of Machine Learning Research*, pages 1640–1649, Naha, Okinawa, Japan, 2019. PMLR

Schaechtle, Ulrich. “InferenceQL: AI for data engineers, without the math.” Talk presented atStrange Loop 2019, September 2019.

Charchut, N. “Implementation of a cross-platform automated Bayesian data modeling system.” *Master's thesis*, Massachusetts Institute of Technology, 2020.

Cusumano-Towner, M. “Gen: A high-level programming platform for probabilistic inference.” *Ph.D.* Thesis, Massachusetts Institute of Technology, 2020.

Saad, F. A.; Rinard, M. C.; and Mansinghka, V. K. “Exact symbolic inference in probabilistic programs via sum-product representations.” *arXiv preprint*, arXiv:2010.03485. 2020. Preprint

Lew, A. K.; Agrawal, M.; Sontag, D.; and Mansinghka, V. K. “PClean: Bayesian Data Cleaning atScale with Domain-Specific Probabilistic Programming.” In *AISTATS 2021: Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130, of *Proceedings of Machine Learning Research*, pages 1927–1935, 2021. PMLR

Schaechtle, U.; Shelby, Z.; Mansinghka, V. “InferenceQL: A SQL-Like probabilistic programminglanguage” poster presented at *PROBPROG* 2020, October 2020.

Saad, F.A.; Rinard, M.C.; Mansinghka, V.K. “SPPL: Probabilistic Programming with Fast ExactSymbolic Inference.” In *PLDI 2021: Proceedings of the 42nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 804–819, 2021. ACM

Schaechtle, U.; Shelby, Z.; Freer, C.; Saad, F.; and Mansinghka, K. “Accelerating inference forInferenceQL via sum-product expressions.” Poster presented at *PROBPROG* 2021, October 2021.

8.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AI	Artificial Intelligence
BLOG	Bayesian Logic
BMI	Body Mass Index
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DNA	deoxyribonucleic acid
DoD	Department of Defense
DSL	Domain Specific Language
ETL	extract, transform, and load
FPGA	Field Programmable Gate Array
FPKM	Fragments Per Kilobase of transcript per Million mapped reads
GPU	Graphics Processing Unit
MCMC	Markov chain Monte Carlo
MCS	Machine Common Sense
MIT	Massachusetts Institute of Technology
ML	Machine Learning
PPAML	Probabilistic Programming for Advancing Machine Learning
RNA	ribonucleic acid
RoI	Return on Investment
SAIL-ON	Science of Artificial Intelligence and Learning for Open-world Novelty
SQL	Structured Query Language
STEM	science, technology, engineering, and mathematics
TPU	Tensor Processing Unit
US	United States
UX	User Experience