# Extending Loom: A Fast, Non-parametric Bayesian Machine Learning Algorithm

Michael Truell

under the direction of
Feras Saad and Dr. Vikash Mansinghka
MIT Probabilistic Computing Group
CSAIL
Massachusetts Institute of Technology

Research Science Institute
August 1, 2017

## Abstract

Loom is a scalable machine learning algorithm that finds meaning in large, high dimensional datasets. Loom infers a latent structure over tabular datasets ranging from tens of rows to millions of rows. At its core, the Loom algorithm is built on Bayesian inference in a hierarchical, nonparametric model. Our contributions to Loom are twofold. First, we have given Loom the ability to perform useful statistical inferences, such as finding dependent features, generating new variables, and estimating similarity between dataset entries. These inference abilities are exposed through the BayesDB probabilistic programming platform, in order to allow non-statisticians to utilize the Loom algorithm for their research. Second, we have experimentally validated the ability of the now-complete Loom algorithm. We have shown that Loom vastly improves on the scalability of its predecessor, the CrossCat machine learning algorithm, with the ability to analyze datasets with millions of cells in a matter of hours. Loom matched the performance of an expert statistical analysis on a diabetic care dataset in a few hours, with no hand-tuning of parameters, and using just 15 lines of source code.



## Summary

We present part of the development "Loom," a highly-scalable machine learning algorithm. Loom ingests big datasets and learns to model their internal dynamics. We have given Loom the ability to generate a number of useful insights using these models. For example, if trained on a macroeconomic dataset, Loom could determine which economic metrics are dependent on one another, even if the dependencies are non-linear and involve multiple variables. Loom could also infer the value of economic indicators; if a country had no recorded measure for unemployment, Loom could provide an estimation of it. Users interact with the Loom algorithm through an easy-to-use, english-like programming language, allowing non-statisticians to use Loom for their research or work. We have shown Loom to be orders of magnitude faster than a similar machine learning algorithm. We have applied Loom to diabetic care analysis, where it matched an expert statistical analysis in under a day.

# 1 Introduction

The analysis of complex, high-dimensional, and sparse databases is a common statistical and scientific problem. To borrow an example from computational biology, the LIGAND enzymatic database [1] is made up hundreds of thousands of rows of data, each containing detailed metrics about the three dimensional structure and interactions of one of ten thousand enzymes. Meaning is hiding in this vast, tabular list of numbers; encoded in it are the fundamental interactions that allow human bodies to function. Statistical analysis tools attempt to tease out meaning in datasets like LIGAND.

One example of a way in which statistical inference systems extract meaning from data is through the discovery of dependent relations between variables in a database. For example, a perfect statistical modeling system would, given a database of economic indicators, be able to self-discover that national GDP growth partially dependent on federal minimum wage policy. The relationship between the two is non-trivial and masked by lots of statistical noise, but a signal is hiding in the data. Another example of a candidate inference query is the prediction of new data points. Returning to our economics example, an ideal statistical system should, given a profile of the economy of a country, be able to estimate the country's unemployment rate.

These sorts of inferences, like dependency detection and data prediction, between variables in datasets has recently become a topic of fervent research. However, they remain statistically and computationally difficult. First, real-world datasets may contain anomalous or missing values. As such, they require cleaning and interpolation before pairwise statistics can be calculated or computational analysis can begin. Second, the modeling assumptions that underlie standard hypothesis testing techniques, such as $R^2$ and HSIC, are often not appropriate, because real-world relationships are nonlinear and multivariate. Third, as the number of database variables grows, it becomes harder to detect true relationships while

suppressing false positives.

The MIT Probcomp Group [2] has proposed a promising approach to statistical inference on sparse and complex datasets that combines probabilistic programming, information theory, and non-parametric Bayesian machine learning algorithms. This system uses the group's Bayesian learning model called CrossCat [3] to group related rows and columns in a database according to a hierarchy of probabilistic mixture models. Using these groupings, CrossCat can model the distribution of each variable in the database. The system is included and wrapped in BayesDB and BQL [4], an SQL-like database and programming language that allow for the easy use of data science utilities.

The CrossCat machine learning system has shown improved sensitivity and specificity over traditional correlation modeling techniques, such as $R^2$ and HSIC, and even state-of-the-art machine learning systems [2]. However, CrossCat only functions on small to medium sized datasets, often those containing less than 10,000 rows and 200 columns. CrossCat's performance issues stem from both implementation inefficiency and algorithmic latency. The CrossCat backend is unoptimized and written entirely in interpreted Python code. Next, its algorithm requires the fitting of at least one statistical model to every pair of values in a dataset.

To address these issues, Dr. Fritz Obermeyer of UC Berkely EECS began and spearheaded the development of Loom, a new machine learning algorithm, with his publishing of the subsampling annealing improvement to the CrossCat algorithm [5]. In addition to mitigating CrossCat's poor scaling ability through the implementation of subsample annealing, Loom further improves upon CrossCat with a parallel, low-level software implementation.

We present the addition inference queries to Loom, including the ability to calculate dependency estimations, predict new data points, and calculate similarity between dataset entries. These inference queries are wrapped in BQL, an easy to use SQL-like programming language, which is amenable to non-computer-scientists. We have benchmarked Loom's in-

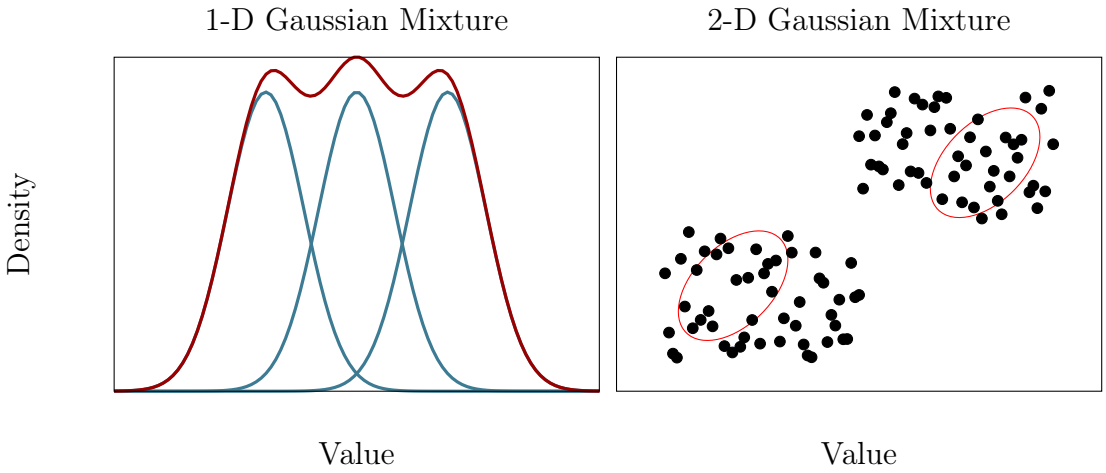


**Figure 1: Left**: A simple univariate Gaussian mixture using three constituent gaussian distributions (blue) to model a combined distribution (red). **Right**: An example of a bivariate mixture model. Two individual univariate gaussian mixture are combined to model the joint distribution of a pair of variables.

ference against its CrossCat predecessor. Finally, to showcase Loom's speed, effectiveness, and ease-of-use, we have applied the system to the analysis of a large-scale diabetic patient dataset, where it matched the performance of an expert statistical analysis in hours and using just 15 lines of source code.

# 2 Background

In this section, we present an introduction to simple probabilistic mixture modeling. We then detail the CrossCat machine learning algorithm and its mixture modeling internals. We next explain the Loom system and its improvements over CrossCat. Finally, we give an introduction to the BayesDB data science platform.

## 2.1 Mixture Models

A mixture model attempts to fit a set (a "mixture") of finite random distributions to the probability distribution of a population of real-life data. One common example of a mixture

model is the gaussian mixture model (shown in Figure 1), which fits a constant number of gaussian distributions to a set of data using a simple hill-climbing optimization method.

However, one cannot know the number of random distributions needed to model a real-world dataset. Under such circumstances, one would use a Dirichlet Process Mixture Model (DPMM) [6]. A DPMM is a non-parametric model that functions by maintaining an infinite distribution over a set of candidate mixture models, each with a number of constituent distributions that starts from 1 and goes to infinity. DPMMs can be used to model many statistical types of data. For example, they may be paired with Gaussian Mixtures to model real numbers, Gamma-Poisson Mixtures to model integers, or Dirichlet-Discrete Mixtures to model categorical data.

## 2.2 CrossCat

CrossCat is a machine learning algorithm for analyzing high-dimensional data tables. At its core, CrossCat groups a dataset, both rows and columns, into multiple non-overlapping clusters of related variables. A visualization of one of these groupings, on a hypothetical dataset of animal characteristics, is shown in Figure 2. The groupings that CrossCat infers have been shown to be consistent with accepted findings and common-sense knowledge in multiple domains [3].

To learn its column and row clusters, CrossCat employs a hierarchy of DPMMs. One DPMM is used to partition the columns of a datset into clusters. The daughter mixtures of this column DPMMs are themselves DPMMs, and each partitions the rows of the dataset within a specific column cluster. The daughter mixtures of row DPMMs are simple mixture models, with the type of the mixture changing according to the statistical type of the data being modeled. For example, integer data is modeled with a Gamma-Poisson distribution and real numbered data is modeled with a Gaussian distribution.

Inference calculations, like predictive accuracy and column dependency, may be derived


**Figure 2:** Figure from [3]. **Top**: A visualization of CrossCat's clustering on an example dataset of animal characteristics. Animal names are on the $y$-axis, and animal features are on the $x$-axis. A green point indicates that the animal at said $y$ coordinate does have the feature on said $x$ coordinate; for example, the square at the top left of the figure indicates that an "Alligator" has the "bites" characteristic. **Bottom**: A visualization of CrossCat's learned clusters on the animal dataset. The algorithm has partitioned the features (the "columns" or "labels" of the dataset) into three, large related blocks (A-C). Groups of related animals within each of these three blocks are denoted by black bold lines.

from CrossCat's learned groupings. These are competitive with state-of-the-art generative, discriminative, and model-free algorithms [2, 7, 3].

## 2.3 Loom

Loom is an extension of the CrossCat machine learning algorithm. Just like CrossCat, it aggregates the columns of a dataset into related groupings, which may be analyzed in order to extract useful information about the dataset. Loom improves on the CrossCat algorithm with better scaling and drastically lower inference runtimes on large datasets. It achieves this speed up through the introduction of subsample annealing and the leveraging of superior software design, as enumerated below.

### 2.3.1 Subsample annealing

Subsample annealing (SA) is an adaptation of the simulated annealing algorithm to nonparametric clustering proposed by [5]. Instead of addressing a whole dataset at once, SA learns the latent structure of a subpopulation of a dataset. This subsample progressively grows as analysis continues. SA speeds up parts of the Loom inference pipeline from $N^2 \rightarrow N$ and others from $\exp(N) \rightarrow N$, where $N$ is data size. Empirically, SA outperforms CrossCat's naive Gibbs sampling in accuracy-per-wallclock time, while sacrificing maximum accuracy [5]. The addition of SA allows Loom scale to larger datasets and deeper hierarchical models than the CrossCat system.

### 2.3.2 Software Engineering

Much of Loom's speed improvement over CrossCat comes from superior software design. Most of this can be attributed to many low-level, individual architecture decisions. However, a few high-level points may be highlighted. First, the Loom system is parallel at its core. Loom splits up all jobs between separate processes using an intelligent batching system. Data

structures are stored and represented as protobuffers using the Google Protobuf library, and thus, they are quickly accessible to all processes. A second reason is language and library choice. CrossCat was implemented in Python and depends on various large, slow libraries. However, Fritz Obermeyer and a team of computer scientists from Navia Systems and UC Berkeley wrote much of Loom's clustering code from scratch in C/C++.

### 2.4 BayesDB

The BayesDB project aims to allow normal computer users to query the probable implications of their data and make inference queries as easily as querying the data itself [4]. BayesDB wraps and aggregates many machine learning models, such as the CrossCat model, in a common, well-designed, and SQL-like programming language. BayesDB is currently used by a small, closed group of NGOs, governmental departments, and academic labs. In just hours to days, BayesDB users have attained better results than those that trained data scientists achieved over the course of weeks accross benchmarks such as flu data analysis and weather modeling.

## 3 Materials and Methods

We extend the Loom machine learning system, adding to it the ability to perform inference queries such as the computation of column dependency, predictive relevance, simulation, etc. These inference queries are wrapped in a BayesDB/BQL programming language frontend.

### 3.1 Addition of Inference Abilities

Loom previously had been able to group related columns and rows in a dataset, learning the characteristic structure of CrossCat detailed in previous sections. However, it lacked the full ability to extract useful information from these groupings. As such, this work presents the

addition of a set of generally applicable inference queries to Loom. These are highlighted in the sections below.

### 3.1.1 Detecting Dependencies Between Columns

To detect dependencies between a pair of columns (variables) $a$ and $b$ from a dataset $X$, it is natural to use a Monte Carlo estimate of the probability that the two variables were grouped into the same Loom column cluster $C$, since this would imply that the two variables are closely related. To calculate this, we may initialize a set of Loom models $L$ of size $N$, which are indexed by $n$. Each model will have a uniquely seeded pseudo-random-number-generator. Thus, each will construct a (slightly) different set of dataset partitions. We can then compute the dependence between $a$ and $b$ by averaging over $L$ like so:

$$\begin{aligned} Pr[C_a = C_b|\mathbf{X}] &\approx \frac{1}{N}\sum_n Pr[C_a^n = C_b^n|L_n] \\ &= \frac{\#(\{n|C_a = C_b\})}{N}. \end{aligned}$$

### 3.1.2 Estimating Row Similarity

Exploratory analyses often make use of a "similarity" function defined over a pair of rows, $x$ and $y$. Such a function quantifies the distance between two rows in a dataset. In the mixture model literature, the similarity problem is often formalized as the probability that two rows were generated from the same underlying statistical model.

The Loom similarity function may be seen as roughly equivalent to the column dependency function above, but with the important distinction of operating on rows instead of columns. Thus, instead of counting the number of column clusters shared by a pair of columns, we count the number of row clusters of which $x$ and $y$ are members. A row cluster

is denoted by $R$ and is indexed by $n$, the index of a single Loom model, and $i$, the index of one of a total of $I$ column clusters for a single Loom model. To compute a similarity estimate, we count and normalize the number of row clusters that $x$ and $y$ share, like so:

$$\begin{aligned} Pr[R_a = R_b|\mathbf{X}] &\approx \frac{1}{N}\sum_n \frac{1}{CC}\sum_{cc} Pr[R^n_{i_a} = R^n_{i_b}|L_n] \\ &= \frac{\#(\{n|R_a = R_b\})}{N * I}. \end{aligned}$$

### 3.1.3 Predicting New Data Points

Loom was given the ability to predict new row values, ranging from the interpolation of a singular missing value to the simulation of whole new rows of data. Formally, Loom must calculate a set of missing row values $\hat{x}_t$ given a set of conditional row values $x_c$. To generate our predictions, we must formally simulate:

$$\{\hat{x}_{t_i}\} \sim p(\{X_{t_i}\}|\{X_{c_i} = x_{c_i}\}, \mathbf{X}).$$

This prediction can be done in two, high-level steps. First, for each column cluster of each Loom model, we must estimate the row cluster to which our incomplete row of conditional values belongs. To determine this, we can sample the conditional density of Loom's outermost DPMM. We must then simulate our missing row values. To do so, we sample from the conditional density of the inner DPMM for our inferred row cluster.

### 3.1.4 Ranking Rows By Relevance

Predictive relevance is a candidate search ranking function for tabular data [7]. The predictive relevance of row $x$ to a query row $y$ in the context of column $c$ measures the relevance of $x$ in formulating predictions about $y$ in the $c$ column. From an information theoretic standpoint,

predictive relevance is equivalent to the posterior probability that the mutual information between $x$ and $y$ in the context of $c$ is non-zero.

Predictive relevance may be elegantly derived from Loom's groupings. A row cluster is denoted by $R$ and is indexed by $n$, the index of a single Loom model. To compute predictive relevance, we count and normalize the number of row clusters that $x$ and $y$ share accross models in the column cluster of column $c$, like so:

$$\begin{aligned} Pr[R_a = R_b|\mathbf{X}] &\approx \frac{1}{N}\sum_n Pr[R_a^n = R_b^n|L_n] \\ &= \frac{\#(\{n|R_a = R_b\})}{N}. \end{aligned}$$

## 3.2 BayesDB Frontend

We wrap Loom's inference queries in BayesDB, an easy to use SQL-like database. The accessibility of BayesDB will hopefully allow professionals and scientists to easily leverage the power of Loom. This will hopefully our work's impact to extend past our respective field and the specific applications presented by this paper to many branches of science and industry.

### 3.2.1 BQL Grammar

The Loom system was fully integrated into the grammar of the Bayesian Query Language, BayesDB's SQL-like interpreted data manipulation language. An example of a full BQL workflow for ingesting data and computing dependency estimations using Loom for the Gapminder economic database is show in Listing 1.

```
-- Ingest CSV file
CREATE TABLE "sql_table" FROM 'resources/gapminder.csv';
-- Create analysis population
CREATE POPULATION "population" FOR "sql_table" WITH SCHEMA (
    GUESS STATTYPES FOR (*);
);
-- Create analysis schema for gapminder dataset
CREATE ANALYSIS SCHEMA "loom_model" FOR "population" USING loom;
-- Analyze gapminder dataset: i.e. compute dependency values etc
INITIALIZE 1 ANALYSIS FOR "loom_model";
-- Calculate Dependencies
ESTIMATE COLUMN DEPENDENCY FROM PAIRWISE VALUES OF "loom_model";
```

**Listing 1:** Ingesting the Gapminder dataset [8] and computing pairwise dependency values on its columns in just seven lines of source code. First, the dataset is imported from a csv file. Next, it is converted from a SQL table to a BQL population. Loom is then initialized and is made to perform an analysis of the Gapminder dataset. Finally, we query loom for the dependency values of the columns of this dataset (these are economic indicators, such as gross exports and GDP).

### 3.2.2 Interfacing with CrossCat

In addition to integration into the BQL grammer, Loom interfaces with MIT Probcomp's existing implementation of CrossCat. Specifically, a BayesDB user is given the ability to convert a Loom model to a CrossCat model after data ingestion and inference. This is necessary because one drawback of the Loom system is its high runtime on a few forms of inference queries (see Speed Comparison in Results). Thus, BayesDB users can ingest and partition a large dataset using Loom, convert Loom to CrossCat, and perform inference queries. The conversion algorithm and code are lightweight, executing in sublinear time.

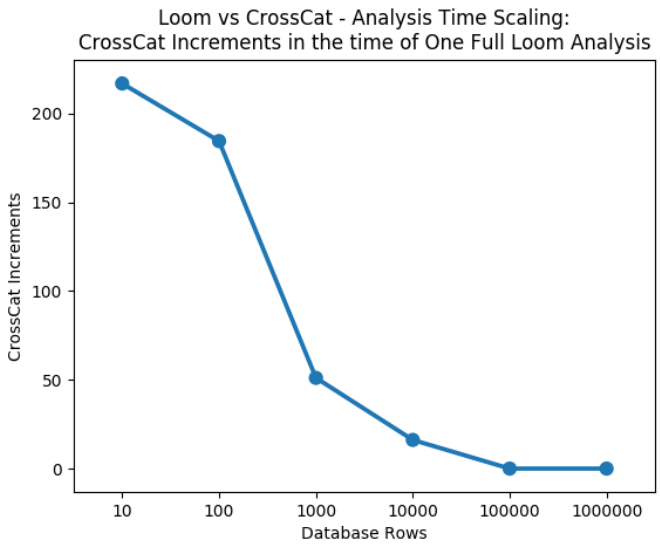


**Figure 3:** This graph shows the number of analysis increments that CrossCat can perform during the same time as one full Loom analysis. One Loom iteration is roughly comparable to hundreds of CrossCat increments. CrossCat can perform many hundreds of increments on datasets of $< 1000$ rows. At 1,000 rows, however, CrossCat can only perform 50 increments. On datasets of 10,000 to 1,000,000 rows, CrossCat cannot finish one increment during the time of a complete Loom analysis. All datasets were synthetic and had 32 columns.

# 4 Results

We have shown that Loom vastly improves on the scalability of the CrossCat machine learning system, rendering analysis of datasets of up to at least 1,000,000 million rows feasible. In addition, Loom performs many inference queries faster than the CrossCat system. Finally, Loom's potential was demonstrated on a dataset of diabetic hospitalizations, on which Loom matched the performance of an expert statistical analysis in just 15 lines of code and with no parameter tuning.

## 4.1 Speed Comparison

Loom and CrossCat perform two distinct types of computation. The first is **analysis** on a dataset, where Loom or CrossCat learns its partitioning of a datset. The second is **query-**

**ing**, when inference queries, such as column dependency detection, are performed. During analysis, Loom or CrossCat fits its mixture models to a dataset and constructs its partition of the rows and columns of said datset. Once analysis has been run, Loom or CrossCat may perform the inference queries enumerated earlier in this paper (predict, dependency, similarity, etc.).

It is important to note that CrossCat and Loom block their analysis in very different ways. CrossCat performs short, iterative passes (or "increments") over a dataset, slightly improving the fit of its DPMMs and the quality of its column and row partitions with each run. However, due to the algorithmic construction of subsample annealing, Loom performs its analysis in one large annealing step. Thus, one Loom analysis is not comparable to one CrossCat increment, making it difficult to compare the analysis runtimes of the two. A rough heuristic is 1 Loom analysis $\approx$ 100-1,000 CrossCat increments.

The analysis runtimes of Loom and CrossCat are compared in Figure 3. Loom scales vastly better than the CrossCat machine learning algorithm. Loom is able to finish a full analysis of datasets ranging from T rows to 1,000,000 rows before CrossCat could even perform one of its increments. One should note that all Loom analyses completed in less than 48 hours. However, single increments of CrossCat on the one million dataset did not finish within several days and, thus, were never run to completion.

The query speeds of Loom and CrossCat are compared in Figure 4. Loom shows significantly better scaling on big datasets and complex inference queries, though is slower on simple, repeated queries.

## 4.2 Applications to Diabetic Care

To test Loom on in a real-life context, we attempted to replicate a study of national diabetic care [9]. The authors of said study assembled a dataset of 100,000 hospitalizations of diabetic patients from 1999 to 2008. Information recorded about each hospital stay included the

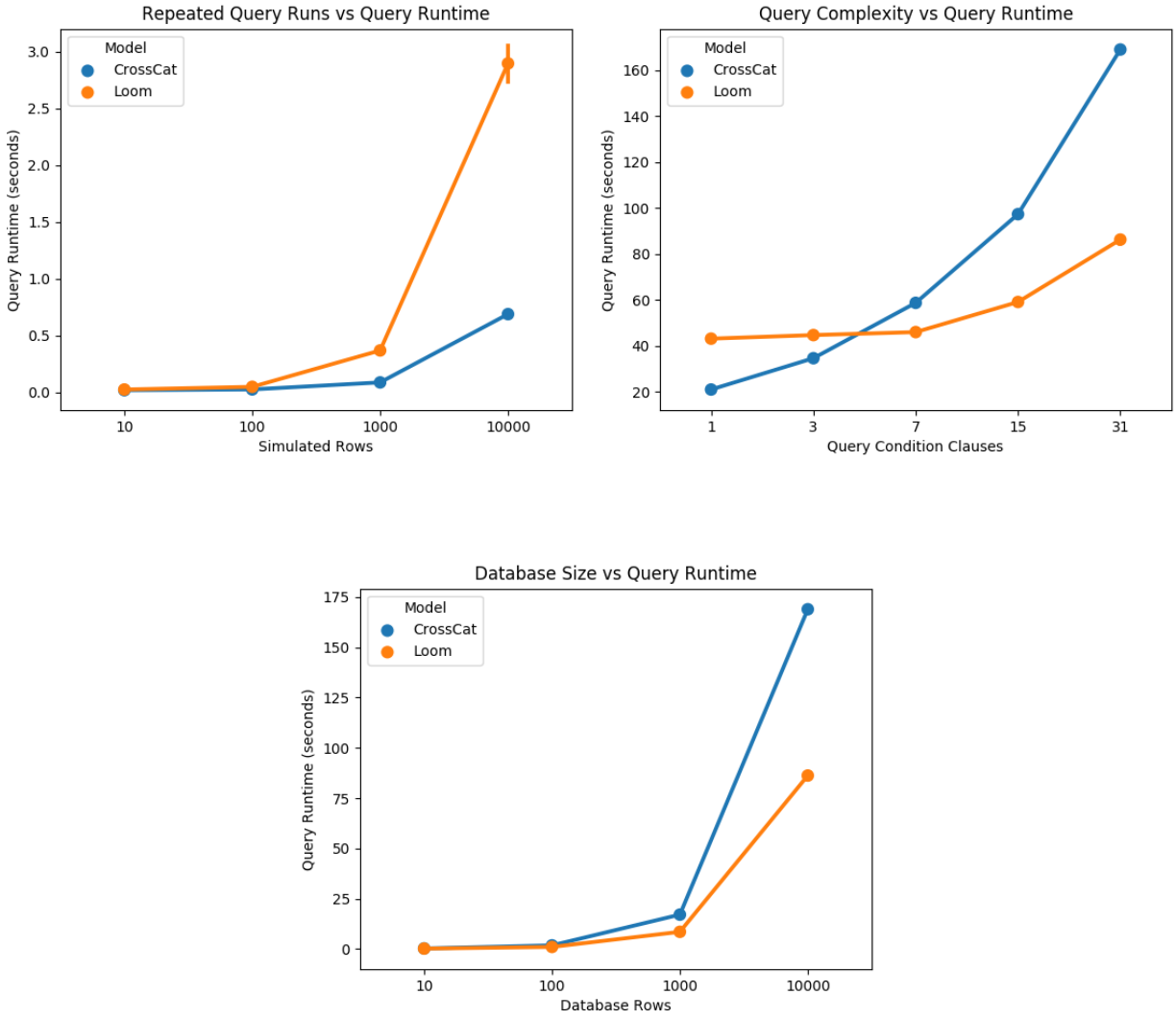


**Figure 4:** An evaluation of the inference query speed of Loom and CrossCat. Loom shows significantly better scaling (lower runtime) on big datasets and complex inference queries, though is slower on simple, repeated queries. **Top Left**: CrossCat and Loom are repeatedly asked to predict new dataset rows. **Top Right**: CrossCat and Loom are asked to predict the likelihood of a hypothetical data entry occuring, and the number of prior conditions (aka. the complexity of the query) is varied. **Bottom**: CrossCat and Loom are again asked to predict the likelihood of a hypothetical data entry occuring, and the size of the dataset is varied. All datasets had 32 columns and were synthetically generated.

patient's current medications, the results of a host of blood tests, demographic information, and future medical incidents. The authors analyzed this dataset to determine the relationship between patient readmit rates and the administration of an HbA1c blood test, a measure of glycated hemoglobin which is used as a proxy for one's three-month average blood plasma glucose concentration.

Strack's analysis required extensive preprocessing and variable control. The authors enlisted a set of "clinical experts" to perform feature selection and prune confounding data entries. They fit a multivariate logit regression model to different partitions of the dataset and controlled for a set of hand-picked variables, including age, primary diagnosis, and the specialty of the admitting doctor. A description of this analysis and its parametrization occupies six, full pages of their paper. In conclusion, they found that, though only administered to 18% of hospitalized diabetics, the HbA1c test significantly lowered early readmission rates, suggesting it should be leveraged much more widely by physicians.

We have applied Loom to the above dataset in order to evaluate the algorithm against an typical data analysis performed by computer scientists and domain experts. Loom requires no preprocessing of the data, no human-tuned parameter selection, and just fifteen lines of BQL source (a listing of this source is included in the appendix). Loom's analysis of the dataset took about six hours for an ensemble of 64 models on a 64 core server.

In accordance with the expert analysis [9], Loom found a significant coupling between the administration of an HbA1c test and readmission rates, assigning the pair a dependency value of 0.8413; dependency values range from 0 to 1, with a value of 1 implying full dependence. A full heatmap of dependence values is shown in Figure 5. This result suggests that Loom is competitive with expert, domain-specific statistical analysis, while requiring comparatively little time, and effort.

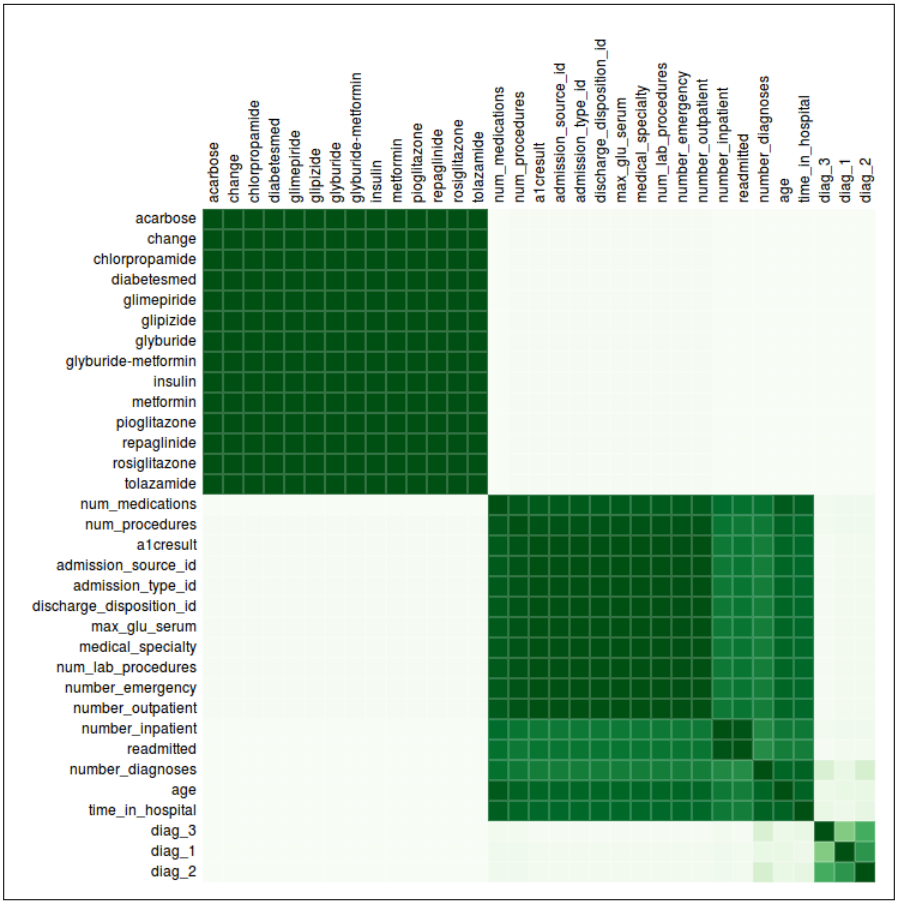


**Figure 5:** A heat map of the pairwise column dependency values that Loom calculated on the diabetes dataset. Green is dependent, while white is independent. **Top Left**: Loom learns to group all of the medications in one large dependent block. This agrees with our intuition; the medications that one may take are restricted by the set that one is already taking, and medications that target the same condition may be commonly assigned together. **Bottom Right**: Loom also learns a dependent relationship between preliminary (diag_1), secondary (diag_2), and tertiary (diag_3) diagnoses.

# 5 Conclusion

A host of inference queries, such as finding dependent features, generating new variables, and estimating similarity between dataset entries, were added to the Loom machine learning algorithm. These inferences are wrapped in the BQL programming language and BayesDB, allowing non-experts to easily interact with the Loom machine learning algorithm. The scaling ability of the Loom algorithm was benchmarked. Loom was shown to successfully ingest datasets of up to one million rows in under a day, while CrossCat could only scale to datasets of ten thousand rows or fewer. Loom applied a large dataset of diabetic patient hospitalizations. The algorithm matched the performance of an expert statistical analysis in a few hours, with no hand-tuning of parameters, and using just 15 lines of source code.

# 6 Acknowledgments

I would like to thank my mentor Feras Saad, my supervisor Dr. Mansinghka, and the MIT Probcomp group for being welcoming teachers and hosts. I would like to thank Peter Lu for his helpful direction and tutelage. I would like to acknowledge the CEE and the RSI for providing me with the opportunity to perform this research and MIT for hosting me. I would like thank my counselors and peers for making RSI an unforgettable experience. Finally, I would like to acknowledge my many sponsors for supporting this research internship, including the Department of Defense, Ansatz Capital, Two Sigma, BEST, Mr. Christopher Chang, Mr. John Yochelson, Andreas Fibig, Dr. Leighton Symons, Dr. Mark Epstein, and Dr. Elyse Rafal.

# References


[1] S. Goto, T. Nishioka, and M. Kanehisa. LIGAND: chemical database for enzyme reactions. *Bioinformatics (Oxford, England)*, 14(7):591–599, 1998.

[2] F. Saad and V. Mansinghka. Detecting dependencies in sparse, multivariate databases using probabilistic programming and non-parametric bayes. In *Artificial Intelligence and Statistics*, pages 632–641, 2017.

[3] V. Mansinghka, P. Shafto, E. Jonas, C. Petschulat, M. Gasner, and J. B. Tenenbaum. CrossCat: A fully bayesian nonparametric method for analyzing heterogeneous, high dimensional data. *arXiv preprint arXiv:1512.01272*, 2015.

[4] V. Mansinghka, R. Tibbetts, J. Baxter, P. Shafto, and B. Eaves. BayesDB: A probabilistic programming system for querying the probable implications of data. *arXiv preprint arXiv:1512.05006*, 2015.

[5] F. Obermeyer, J. Glidden, and E. Jonas. Scaling nonparametric bayesian inference via subsample-annealing. volume 33, 2014.

[6] C. E. Rasmussen. The infinite gaussian mixture model. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 554–560, Cambridge, MA, USA, 1999. MIT Press.

[7] F. Saad, L. Casarsa, and V. Mansinghka. Probabilistic search for structured data via probabilistic programming and nonparametric bayes. *arXiv preprint arXiv:1704.01087*, 2017.

[8] H. Rosling, A. Rosling Ronnlund, and H. Rosling. Gapminder: unveiling the beauty of statistics for a fact based world view. *URL https://www. gapminder. org/data*, 2010.

[9] B. Strack, J. P. DeShazo, C. Gennings, J. L. Olmo, S. Ventura, K. J. Cios, and J. N. Clore. Impact of hba1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records. *BioMed research international*, 2014, 2014.

[10] F. Saad and V. K. Mansinghka. A probabilistic programming approach to probabilistic data analysis. In *Advances in Neural Information Processing Systems*, pages 2011–2019, 2016.

[11] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

# Appendix A Experimental Details

All tests were run on an Debian-based server with 64 CPU cores, 500GB RAM, and ample storage space.

For the analysis benchmark, Loom and CrossCat were run on a synthetic dataset which was generated to have 1 column cluster, 32 columns, and a number of rows from 10 to 1,000,000.

For the query benchmarks, Loom and CrossCat were also run on synthetic data with 1 column cluster and 32 columns. The probability density and data prediction queries were chosen as representative queries, as these are two of the most generally applicable inferences available to Loom and BayesDB.

The source listing for our diabetes analysis is shown in Listing 2. Senseless IDs (payer_code, patient_nbr) and very sparse features ignored (race, gender, weight) were ignored by Loom. After Loom's initial analysis annealing, the system ran 1000 extra passes over the data.

```
CREATE TABLE "dia" FROM 'resources/diabetes.csv';
CREATE POPULATION "dia_pop" FOR "dia" WITH SCHEMA (
    GUESS STATTYPES FOR (*);
    MODEL
    "diag_1",
    "diag_2",
    "diag_3"
    AS unboundedcategorical;
    IGNORE race, gender, weight, payer_code, patient_nbr;
);
CREATE ANALYSIS SCHEMA "dia_meta" FOR "dia_pop" USING loom;
INITIALIZE 64 ANALYSIS IF NOT EXISTS FOR "dia_meta";
ANALYZE "dia_meta" FOR 1 ITERATIONS WAIT;
ESTIMATE DEPENDENCE PROBABILITY FROM PAIRWISE VARIABLES OF dia_pop;
```

**Listing 2:** The full dependency analysis source for our analysis of a dataset of diabetic hospitalizations from 1999-2008 [9]. The code is at most 15 lines long.