Accelerating inference for InferenceQL via sum-product expressions

Ulrich Schaechtle Zane Shelby Cameron Freer Feras Saad Vikash K. Mansinghka

MIT Probabilistic Computing Project

What is InferenceQL?

InferenceQL is a SQL-like language for querying probabilistic programs modeling tabular data. InferenceQL insulates users from implementation details of the underlying model by treating probabilistic model programs as black boxes that satisfy a common API. Like probabilistic databases [1], InferenceQL aims to "[find] valuable facts in imprecise data" [2], but whereas probabilistic databases "make uncertainty a first class citizen" [2], InferenceQL makes probabilistic models, given by probabilistic programs, first class objects that can be directly queried in light of data.

Why optimize InferenceQL?

Fast, exact marginalization and conditioning has recently been introduced for a broad class of probabilistic programs that can be compiled into sum-product expressions [3]. We demonstrate that this work can be used to efficiently build and query probabilistic programs that model tabular data. The benefits are twofold: (i) using fast inference leads to more practical workflows for iterative queries, and (ii) the exact inference algorithms have more predictable query runtime. We present preliminary results on performance improvements in existing benchmarks for applying fast exact inference via sum-product expressions to generative population models.

Related work

Probabilistic circuits [4, 5] are a general class of probabilistic models that aim to balance expressiveness with tractable inference for querying. This class includes arithmetic circuits [6], sum-product networks [7], and their generalization to sum-product expressions [3] which are used by InferenceQL. Probabilistic programming languages have made use of compilation to circuits [8], and some approaches to probabilistic circuits have used symbolic inference [9]. Probabilistic circuits have also been used to build probabilistic models induced by random forests [10], a popular ML approach to modeling tabular data in databases.

Probabilistic databases assign weights to facts (grounded atoms of database predicates) in the database [1]. Queries amount to computing the probability of a Boolean formulas over those facts. The weight of a relation in a table needs to be known ahead of time.

Open world probabilistic databases also assign weights to facts, and then create a sum-product network for a given query [11]. In this approach, the probabilities of unknown facts can be assigned any probability value from a default probability interval.

Query plan optimization with via inference aims to speed up costly database queries, e.g. by applying sum-product structure learning to joins in databases [12]. Using an SPN model, [12] speeds up aggregation queries, e.g. estimating an empirical expectation instead of computing the mean over a large table. For approaches using weighted model counting, see [13]. Factorized databases [14] use sums and products to avoid redundant computation in queries.

Extending capabilities of standard databases. Database systems and query languages have been introduced to extend the standard relational algebra with models and predictive modeling capabilities. To model database tables, these systems either come precanned with models or they support specific model classes. They extend relational algebra with functions that deal with imputation [15], time series prediction [16], random data generation [17] and simulation [18].

BayesDB and generic APIs for querying probabilistic programs. InferenceQL is most similar to BayesDB [19] in that it aims to provide a querying API to generic probabilistic programs modeling tabular data. Like InferenceQL, BayesDB requires all models to satisfy the CGPM API [20]. However, InferenceQL introduces further restrictions on compositions to soundly support a broad class of probabilistic programs. The CGPM API itself imposes tighter restrictions than other interfaces for probabilistic programs (e.g. the generative function interface in Gen [21] and the stochastic procedure in Venture [22]).

Platform overview

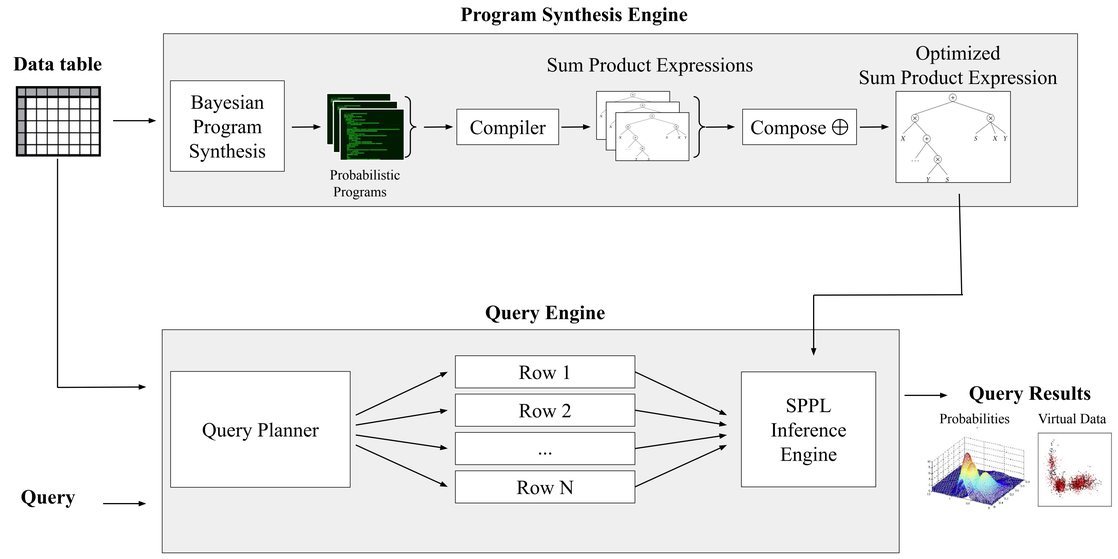


Fig. 1. Overview of the platform implementing InferenceQL. The platform consists of a program synthesis engine as well as a query engine. The synthesis engine takes a data table as input and returns an SPE expression. This SPE expression is read by the query engine. A query is then processed by a query planner that issues multiple SPPL queries to the SPPL synthesis engine.

Example: learning a probabilistic program for a database of Earth satellites

Perigee	Period_minutes	Dry_Mass_kg	Apogee	Launch_Mass_kg	Eccentricity	Power_watts
500.0	94.68	NaN	506.0	3.0	0.00044	NaN
622.0	97.20	NaN	623.0	1700.0	0.00007	NaN
35778.0	1436.10	NaN	35794.0	5000.0	0.00019	NaN
35778.0	1436.06	1650.0	35792.0	3725.0	0.00017	5640.0
19134.0	676.10	NaN	19141.0	1480.0	0.00014	1600.0
35775.0	1436.04	660.0	35795.0	1430.0	0.00024	1200.0
35781.0	1436.21	1503.0	35797.0	3200.0	0.00019	NaN
35785.0	1436.13	800.0	35789.0	1459.0	0.00005	1600.0
35782.0	1436.14	2389.0	35792.0	3659.0	0.00012	9900.0
35770.0	1436.10	700.0	35802.0	1420.0	0.00038	1500.0
1413.0	114.10	NaN	1414.0	700.0	0.00006	1700.0
35781.0	1436.12	2500.0	35792.0	3130.0	0.00013	4600.0
786.0	100.70	22.0	796.0	45.0	0.00070	160.0
1477.0	115.80	NaN	1509.0	280.0	0.00203	300.0
35780.0	1436.08	3010.0	35791.0	3379.0	0.00013	6600.0
538.0	95.40	NaN	540.0	46.0	0.00014	NaN
500.0	94.70	NaN	504.0	2650.0	0.00029	NaN
563.0	704.30	1250.0	39128.0	2400.0	0.73554	NaN
35775.0	1436.07	1955.0	35796.0	4723.0	0.00025	7000.0
1413.0	114.10	350.0	1414.0	450.0	0.00006	1500.0
35786.0	1436.30	NaN	35796.0	3325.0	0.00012	NaN
820.0	101.30	3750.0	821.0	4193.0	0.00007	NaN
35756.0	1436.11	884.0	35817.0	1156.0	0.00072	1240.0
1100.0	107.26	NaN	1100.0	5000.0	0.00000	NaN

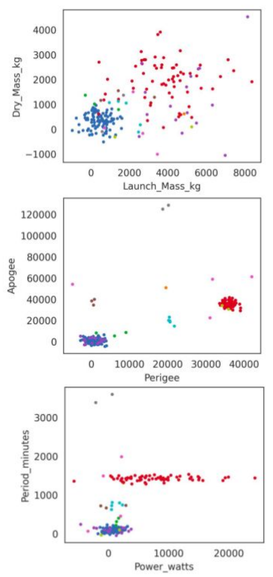
(a) Subset of satellites data table [Union of Concerned Scientists 2016]

```
function view_1_model () {
  var cluster_id = categorical([0.945, 0.02, 0.01, ...]) + 1;
  var ret_val;
  if (cluster_id == 1) {
    ret_val = {
      "Eccentricity": gaussian(0.0020432129206960077, 0.01067912668646529)
    } else if (cluster_id == 2) {
      ret_val = {
        "Eccentricity": gaussian(0.07464751748477876, 0.015180096831659037)
    } else if (cluster_id == 3) {
      ret_val = {
        "Eccentricity": gaussian(0.28533222056967295, 0.01708745553936537)
    ...
  return ret_val;}

function view_2_model () {
  var cluster_id = categorical([0.45, 0.365, 0.01, ...]) + 1;
  var ret_val;
  if (cluster_id == 1) {
    ret_val = {
      "Power_watts": gaussian(870.3285769734457, 877.8033300883731),
      "Launch_Mass_kg": gaussian(442.08158778373314, 528.6317071151509),
      "Dry_Mass_kg": gaussian(362.4518291923288, 321.643115056201),
      "Period_minutes": gaussian(101.67533514957961, 56.020420134225084),
      "Perigee": gaussian(683.494848571412, 1332.796193936696),
      "Apogee": gaussian(747.6838643344822, 2411.9120907168577)};
  } else if (cluster_id == 2) {
    ret_val = {
      "Power_watts": gaussian(7157.582291136565, 4629.0978933649985),
      "Launch_Mass_kg": gaussian(3870.9605891838637, 1417.0856648235301),
      "Dry_Mass_kg": gaussian(1921.3138079297019, 762.0773387715883),
      "Period_minutes": gaussian(1435.6303089573516, 57.13415025152961),
      "Perigee": gaussian(35820.37445538106, 1434.5708880223177),
      "Apogee": gaussian(35701.83498101731, 2548.6013620853966)};
  } else if (cluster_id == 3) {
    ret_val = {
      "Power_watts": gaussian(663.6166769560301, 2240.1933836607705),
      "Launch_Mass_kg": gaussian(4155.624495377741, 1484.9293537892536),
      "Dry_Mass_kg": gaussian(963.0435893006963, 915.5101654965363),
      "Period_minutes": gaussian(24.003429083165 94, 81.81512446233805),
      "Perigee": gaussian(35710.595274948006, 2091.1679968553717),
      "Apogee": gaussian(35706.382689189195, 3696.3766128285724)};}
  ...
  return ret_val; }

function model() {
  var view_1 = view_1_model();
  var view_2 = view_2_model();
  return {...view_1, ...view_2};}
```

(b) Row generator for the satellites data table (JavaScript representation)



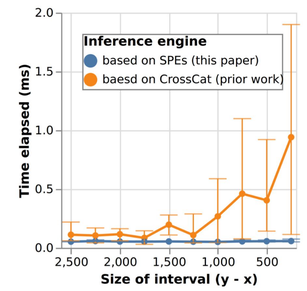
(c) Executions of the probabilistic program

Fig. 2. Probabilistic program in JavaScript, modeling a data table of satellites with an automatically synthesized hierarchical mixture model. The model class over which we are performing inference was introduced in [3]. In this case, a table of satellite data (a) is modeled by a generative program (b), implementing a hierarchical mixture model, an instance of the model class over which we are performing inference. We highlight executions of the program corresponding to each of three clusters (in blue, red, purple) in both the code and scatter-plots (c).

Empirical results

```
SELECT * FROM
    GENERATE Apogee_km
    UNDER model
    CONSTRAINED BY
        Perigee_km > x AND Perigee_km < y
    LIMIT 1
```

(a) Parametric query, where the values of x and y vary, controlling the size of the interval y − x.



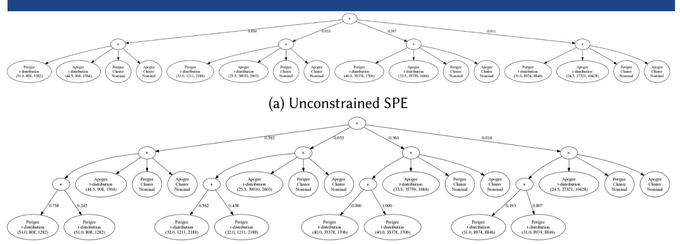
(b) Runtime measurements; error-bars show upper and lower quartile boundaries.

Fig. 3. Accelerated SPE inference is faster and more predictable than CrossCat inference. (a) A parametric query controls the width of a numerical constraint on Perigee_km. (b) The runtimes, expressed in terms of the size of the interval, demonstrate that SPE inference runtime is nearly constant, while CrossCat inference has high variance, especially for longer runtimes.

Table 1. InferenceQL inference runtimes are faster via SPE than via CrossCat. This table shows run-time and variance for queries of the form `SELECT * FROM ([Generate Expression]) LIMIT 100`. Queries were run ten times in InferenceQL for each Generate Expression, with both the SPPL-SPE and CrossCat backends. In every case, SPE was faster, sometimes substantially so.

Generate Expression	Inference Engine: based on SPEs (this paper)	Inference Engine: based on CrossCat (prior work)
`GENERATE Period_minutes UNDER model CONSTRAINED BY Country = "USA" OR Country = "CHINA" OR Country = "multinational"`	**242 ± 0.4**	305 ± 0.8
`GENERATE Period_minutes UNDER model CONSTRAINED BY (Country = "USA" OR Country = "CHINA" OR Country = "multinational") AND Purpose= "Communications"`	**269 ± 5.0**	695 ± 3.1
`GENERATE Period_minutes UNDER model CONSTRAINED BY (Country = "USA" OR Country = "CHINA" OR Country = "multinational") AND Purpose = "Communications" AND Perigee_km > 30000`	**307 ± 14.6**	1730 ± 30.3
`GENERATE Operator_Owner UNDER model CONSTRAINED BY Launch_Site = "Wallops Island Flight Facility"`	**401 ± 2.7**	17217 ± 1307.7
`GENERATE Country, Launch_Vehicle UNDER model CONSTRAINED BY Launch_Site = "Wallops Island Flight Facility" AND Apogee_km < 263038.17`	**412 ± 10.0**	5939 ± 169.4
`GENERATE Type_of_Orbit, Perigee_km, Eccentricity UNDER model CONSTRAINED BY Power_watts > 13150.66 AND Dry_Mass_kg < 3059.18 AND Launch_Mass_kg < 14264.71`	**361 ± 4.0**	616 ± 1.4

How does exact inference work?



(a) Unconstrained SPE

(b) SPE from (a) constrained by the event `Perigee < 1000 OR Perigee > 2000`

Fig. 4. Sum Product Expressions before and after constraining. Constraining a model expressed as an SPE produces another SPE. The constraint re-weights clusters in the networks and adds a sum-rooted subtree to each leaf node that models the constrained variable (Perigee), to deal with the disjoint cases.

Bibliography

[1] Suciu, D., Olteanu, D., Koch, C., & Koch, C. (2011). Probabilistic Databases. Morgan & Claypool Publishers.

[2] Dalvi, N., Ré, C., & Suciu, D. (2009). Probabilistic databases: diamonds in the dirt. Communications of the ACM, 52(7), 86-94.

[3] Saad, F. A., Rinard, M. C., & Mansinghka, V. K. (2021). SPPL: probabilistic programming with fast exact symbolic inference. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation.

[4] Van den Broeck, G., Di Mauro, N., and Vergari, A. (2019) Tractable probabilistic models: Representations, algorithms, learning, and applications. Tutorial at the 35th Conference on Uncertainty in Artificial Intelligence (UAI).

[5] Vergari, A., Choi, Y., Peharz, R., & Van den Broeck, G. (2020). Probabilistic circuits: Representations, inference, learning and applications. Tutorial at the 34th AAAI Conference on Artificial Intelligence.

[6] Darwiche, A. (2003). A differential approach to inference in Bayesian networks. Journal of the ACM (JACM), 50(3), 280-305.

[7] Poon, H., & Domingos, P. (2011). Sum-product networks: A new deep architecture. In 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops) (pp. 689-690).

[8] Holtzen, S., Van den Broeck, G., & Millstein, T. (2020). Scaling exact inference for discrete probabilistic programs. Proceedings of the ACM on Programming Languages, 4(OOPSLA), 1-31.

[9] Kolb, S., Mladenov, M., Sanner, S., Belle, V., & Kersting, K. (2018, July). Efficient Symbolic Integration for Probabilistic Inference. In IJCAI (pp. 5031-5037).

[10] Correia, A. H., Peharz, R., & de Campos, C. (2020). Joints in Random Forests. arXiv preprint arXiv:2006.14937.

[11] Ceylan, I. I., Darwiche, A., & Van den Broeck, G. (2016). Open-world probabilistic databases. In Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning.

[12] Hilprecht, B., Schmidt, A., Kulessa, M., Molina, A., Kersting, K., & Binnig, C. (2020). DeepDB: Learn from Data, not from Queries! Proceedings of the VLDB Endowment, 13(7).

[13] Van den Broeck, G., & Suciu, D. (2015). Query processing on probabilistic data: A survey. Foundations and Trends® in Databases, 7(3-4).

[14] Olteanu, D., & Schleich, M. (2016). Factorized databases. ACM SIGMOD Record, 45(2), 5-16.

[15] Cambronero, J., Feser, J. K., Smith, M. J., & Madden, S. (2017). Query optimization for dynamic imputation. Proceedings of the VLDB Endowment, 10(11).

[16] Agarwal, A., Alomar, A., & Shah, D. (2021). tspDB: Time series predict DB. In NeurIPS 2020 Competition and Demonstration Track (pp. 27-56).

[17] Jampani, R., Xu, F., Wu, M., Perez, L. L., Jermaine, C., & Haas, P. J. (2008). MCDB: a Monte Carlo approach to managing uncertain data. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data.

[18] Farley, S., Brodsky, A., Egge, N., & McDowall, J. (2010). SimQL: Simulation-based decision modeling over stochastic databases. 15th IFIP WG8, 3.

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

[20] Saad, F., & Mansinghka, V. K. (2016). A probabilistic programming approach to probabilistic data analysis. Advances in Neural Information Processing Systems, 29.

[21] Cusumano-Towner, M. F., Saad, F. A., Lew, A. K., & Mansinghka, V. K. (2019). Gen: a general-purpose probabilistic programming system with programmable inference. In Proceedings of the 40th ACM SIGPLAN International Conference on Programming Language Design and Implementation.

[22] Mansinghka, V., Selsam, D., & Perov, Y. (2014). Venture: a higher-order probabilistic programming platform with programmable inference. arXiv preprint arXiv:1404.0099.