

Approximate Query Processing with the Patch Probabilistic Compiler

Zhaoyi Ge
zhaoyi.ge@uwaterloo.ca

1 Motivation and Research Question

As datasets grow, exact query processing becomes increasingly expensive. Approximate query processing (AQP) trades accuracy for efficiency to support fast analytics. Sampling-based systems estimate results from subsets of data, while model-based systems learn predictive models offline. Complex joins and group-by queries remain challenging: sample quality affects accuracy, and evaluating groups independently limits shared computation [2].

BayesAQP [2] takes a model-based approach: it learns Bayesian networks that capture relationships among table attributes. At query time, SQL predicates and aggregates become probabilistic inference tasks, compiled through BTL++ into optimized tensor computations on CPUs or GPUs. Retaining grouping attributes as tensor dimensions allows groups to be evaluated together.

Patch is an unpublished probabilistic language and compiler for describing discrete random variables and computing probabilities from their relationships and observed evidence. Higher-order functions support reusable model components and query operators. The compiler represents computations as tensor networks and chooses variable-elimination steps by cost. It also handles deterministic operations symbolically, avoiding explicit indicator tensors, and executes array programs on NumPy or JAX.

This project will develop an AQP framework using Patch. The central question is how to express relational queries as Patch programs that support efficient inference and composable query operators, and what tradeoffs result in latency, memory use, and approximation accuracy.

2 Proposed Approach

The approach follows BayesAQP’s separation of data modeling from query compilation [2]. Offline, the framework will learn compact probabilistic models of table attributes and collect statistics for joins. At query time, it will translate SQL into Patch programs that estimate results by performing inference over these models. The research focus is this translation and its compilation, using the BayesAQP’s modeling approach as a starting point.

The key idea is to express relational operations as questions about the modeled data distribution. For example, a filtered COUNT can be estimated by computing the probability that a row satisfies the predicate and multiplying by the table size. SUM and AVG additionally use the modeled values of the

aggregated attributes. Joins combine models using join statistics, while GROUP BY retains the grouping attributes so that inference produces estimates for all groups together. These results are approximate because the models summarize the original data.

The main research task is to design a compositional translation from SQL into Patch. I will develop reusable Patch functions for table models, predicates, and aggregates, then investigate how these components can be combined to express complete queries. Higher-order functions will allow operators to accept query-specific predicates and aggregate expressions, so that extending the supported SQL fragment can build on existing components. For example, conditional aggregation such as `SUM(CASE WHEN p THEN revenue ELSE 0 END)` will test whether a predicate and an aggregate can be composed without a separate compilation rule for every combination. I will examine both whether these abstractions preserve the intended query semantics and how they affect generated code.

Patch’s compiler makes the representation of these operators consequential for performance. It turns a program into a tensor network and uses a cost model to choose the order of inference computations, much as a database optimizer chooses an execution plan. Its symbolic treatment of deterministic operations may also avoid large intermediate tensors for predicates and conditional expressions. I will investigate whether the SQL translation exposes these opportunities across composed operators, and whether the resulting NumPy or JAX programs reduce execution time and memory use. This connects the language design question—how to express reusable query operators—with the systems question of how efficiently they execute.

3 Evaluation

I will evaluate aggregates, joins, and grouped queries on the Star Schema Benchmark [1]. The baseline will be BayesAQP and potentially other AQP frameworks.

The primary metrics will be query execution latency and relative error against exact SQL results, measuring whether Patch can answer queries faster while maintaining approximation quality. I will report compilation time separately to show its contribution to the cost of a first query, and measure peak memory use to assess scalability as joins and group counts increase.

4 Scope and Expected Outcome

The project will produce an AQP prototype built on Patch. It will support COUNT, SUM, and AVG queries with filters and GROUP BY. The implementation will include the SQL-to-Patch translator, trained table models, join statistics, and reproducible evaluation scripts. The report will analyze when tensor optimizations reduce inference latency, and whether estimation accuracy is preserved.

References

- [1] Patrick O’Neil, Elizabeth O’Neil, Xuedong Chen, and Stephen Revilak. 2009. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Performance Evaluation and Benchmarking*. Springer, Berlin, Heidelberg, 237–252. doi:10.1007/978-3-642-10424-4_17
- [2] Jingwen Pan, James Cheney, and Amir Shaikhha. 2026. A Backend-Agnostic Compiler for Approximate Query Processing with Probabilistic Tensor Algebra. *Proceedings of the ACM on Management of Data* 4, 3, Article 126 (2026), 25 pages. doi:10.1145/3802003