DataSwift: Smart Choices for Safe Query Optimization

Raahim Lone
Independent
raahimlone@gmail.com

Abstract

Learned query optimizers struggle to generalize, causing performance regressions for a subset of queries. To address this, DataSwift is introduced, a hint-recommendation framework that integrates LLM-derived SQL embeddings, GNN-encoded plan representations, a similarity-threshold memory cache, and Thompson-sampling bandit exploration. Incoming queries are embedded to recall proven hints; a low-rank inductive matrix completion model predicts expected latency. Validated hints are cached and the bandit down-weights any hints inducing slowdowns. On the combined JOB benchmarks, DataSwift incurs only a **0.7%** regression rate with zero catastrophic regressions and delivers a **1.4**× improvement on the 5% slowest queries. Thus, DataSwift provides performance gains without sacrificing safety.

1 Introduction

Modern databases use query optimizers to generate SQL query execution plans, typically relying on heuristics and cost models. Recently, researchers have introduced machine learning–based optimizers that guide this process by suggesting SQL query hints. Hints are directives that influence join order, memory allocation, or indexing choices [1]. While these learned hints can reduce query latency by steering the optimizer toward more efficient strategies, they also risk *performance regressions*, where queries that once ran quickly under the traditional optimizer become slower under the learned model.

Empirical studies confirm that performance regressions occur frequently in query workloads [2, 3]. To counteract this, a growing body of work has sought to mitigate regressions by supplementing learned cost models with fallback mechanisms:

- **Offline plan validation.** Systems like LimeQO [4, 5] and AutoSteer [6] pre-execute candidate plans to guarantee safety, but rely on repeated query patterns to reduce latency.
- **Uncertainty modeling.** Lero [7] and RoQ [8] estimate cost distributions, reducing regressions but requiring expensive retraining and often misestimate variance.
- **Adaptive Recall and Bandits.** Nearest-neighbor methods [3] and bandit learning [1] adapt online, sometimes aided by LLM embeddings [9], but incur lookup overhead and unstable exploration.
- **Delta-based filters.** PerfGuard [2] and Eraser [3] predict regressions via plan deltas or runtime checks, but remain limited to offline settings or coarse heuristics.

Together, these works suggest that a regression-free learned optimizer must combine (1) expressive query representations for generalization, (2) memory of past outcomes to avoid repeated mistakes, and (3) balanced exploration under uncertainty.

To address these needs, DataSwift is proposed. DataSwift focuses on the subset that causes the most harm: **major regressions**, defined as slowdowns that add $\geq$ **5 s** over the default plan (see Preliminary Experiments). The contributions are as follows:

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025).

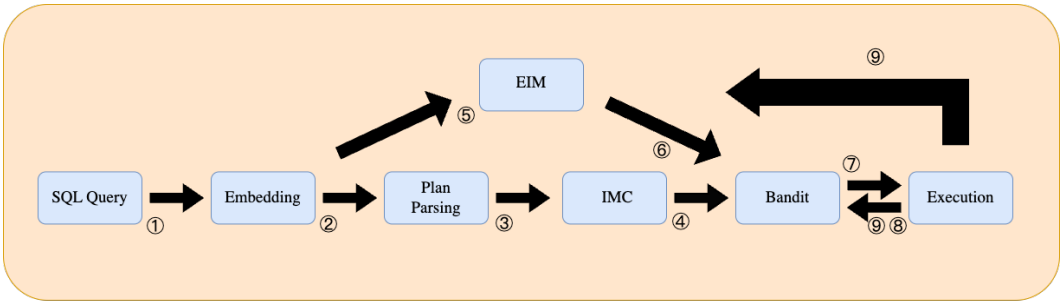


Figure 1: This figure demonstrates DataSwift, which takes incoming SQL queries, parses their plans, and uses an IMC model to predict hint performance. IMC training and Embedding-Indexed Memory (EIM) maintenance are conducted offline, while the rest of the workflow is conducted online.

1. **Inductive matrix completion (IMC) for hint prediction.** IMC predicts hint latencies with uncertainty, enabling principled uncertainty-aware ranking.
2. **Hybrid query representations.** Each query is encoded with both LLM-derived semantic embeddings and GNN-based plan encodings, allowing generalization beyond template repetition.
3. **Embedding-Indexed Memory (EIM).** A lightweight cache of empirically validated query–hint outcomes enables safe reuse of past results without expensive nearest-neighbor search.
4. **Bandit-driven adaptation with safe fallback.** A Thompson-sampling bandit integrates IMC's uncertainty estimates, balancing exploration and exploitation.

Why this generalizes. Hybrid representations capture both *what* a query asks (SQL semantics) and *how* it executes (plan topology). IMC maps queries and hints into a shared low-rank space, enabling predictions for unseen query–hint pairs with uncertainty. Online adaptation, using the default plan as a bandit arm and recalling validated outcomes from EIM, prevents repeated mistakes.

2 System Overview

Upon receiving a raw SQL query (1), DataSwift generates a fixed-length SQL embedding (2) using a pretrained Sentence-Transformers model, and parses the query into a logical operator DAG for a GNN-based (graph neural network; GNN) plan embedding (3). These embeddings are fed into the IMC predictor (4), which formulates hint selection as an IMC problem to output both a mean latency estimate and an uncertainty score for each candidate hint. If the predicted latency improvement is too small or too large, the hint is skipped or avoided. Before applying the top IMC suggestion, DataSwift queries the Embedding-Indexed Memory (EIM) (5), a vector index of past queries indexed by SQL embeddings, to identify any validated hint (6) as an additional candidate. Next, the Bandit Selector (7) treats the IMC's top suggestion, any EIM-returned hint, and the default (no-hint) option as arms in a multi-armed bandit. It samples from each arm's posterior to choose a hint. That chosen hint is then applied and sent to the execution engine (8). Finally, the observed execution latency updates the bandit's parameters and is stored back in EIM for future lookups (9).

2.1 SQL Embedding

The SQL embedding module maps raw SQL text into a semantic vector using Sentence-Transformers. Incoming queries are encoded and compared via Euclidean distance against stored embeddings in an indexed memory, enabling retrieval of proven hints for similar queries. This avoids the variance issues of probabilistic cost models and eliminates the need for offline validation on each new template.

2.2 Plan Graph Encoding and IMC Prediction

Each query plan is parsed into a DAG and encoded by a GNN into a 512-d structural embedding $\mathbf{z}_{\text{struct}} \in \mathbb{R}^{512}$ [10]. In parallel, the SQL text is embedded via a Sentence-Transformers model,

reduced to 120 dimensions with PCA, yielding $\mathbf{z}_{\text{text}} \in \mathbb{R}^{120}$. Concatenating both gives

$$\mathbf{x} = \begin{bmatrix} \mathbf{z}_{\text{struct}} \\ \mathbf{z}_{\text{text}} \end{bmatrix} \in \mathbb{R}^{632}.$$

Alignment between modalities is handled by IMC's learned projections U and V that map queries and hints into a shared low-rank space.

The IMC module maps $\mathbf{x}$ and each candidate hint embedding $\mathbf{h}_h \in \mathbb{R}^{d_h}$ (49 hints from [1]) into a shared rank-r space using

$$\mu_{q,h} = (\mathbf{x}U)^\top (\mathbf{h}_h V) + b_z + b_h,$$

where $U \in \mathbb{R}^{632\times r}, V \in \mathbb{R}^{d_h \times r}$, and b_z + b_h are bias vectors. To capture predictive confidence, $\left[\mathbf{u}_q; \mathbf{v}_h; |\mathbf{u}_q - \mathbf{v}_h|\right]$ is formed and passed through a linear layer, producing an uncertainty score $\sigma_{q,h}$.

Training follows a variance-aware IMC objective [11], restricting supervision to pairs (q, h) where prediction and ground truth align within 2.5σ. The loss is the negative log-likelihood:

$$\mathcal{L} = \frac{1}{|\mathcal{I}|} \sum_{(q,h)\in\mathcal{I}} \left(\frac{1}{2} \log \sigma_{q,h} + \frac{(y_{q,h} - \mu_{q,h})^2}{2\sigma_{q,h}} \right).$$

At inference, the model outputs expected latency and variance $(\mu_{q,h}, \sigma_{q,h})$ for each hint. Predictions are ranked by $\mu_{q,h}$, with $1/\sigma_{q,h}$ serving as a calibrated confidence. This design generalizes to unseen query–hint pairs, avoids outlier domination, and provides principled confidence estimates absent in static plan-delta methods.

2.3 Embedding-Indexed Memory (EIM)

EIM provides a case-based safety net by caching queries with their empirically validated best hints. It stores tuples

$$(\mathbf{e}_i, h_i^*, n_i, b_i),$$

where $\mathbf{e}_i$ is a normalized query embedding, h_i^* its best-observed hint, n_i the execution count, and b_i the number of failures. At inference, a Faiss L_2 index retrieves the top-k neighbors within radius τ. Neighbors with failure rates above 40% are discarded. The radius adapts after each query: $\tau \leftarrow \tau \times 0.92$ on success or $\tau \times 1.07$ on failure. Entries are updated online when a recalled hint outperforms both the default and the prior stored hint.

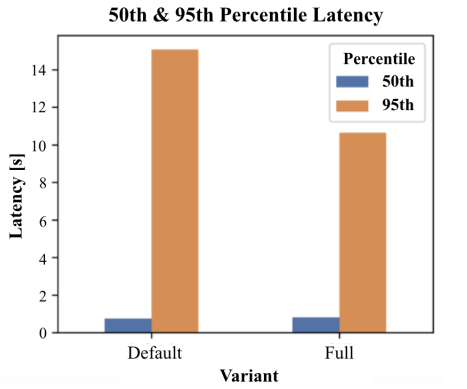
EIM handles (i) **cold-start queries** with high IMC uncertainty and (ii) **recurring patterns** where IMC mispredicts. If a recalled hint outperforms both the default and prior stored hint, the entry is updated; memory is bounded with FIFO eviction. By grounding recall on embedding similarity and validated outcomes, EIM avoids the heavy cost of offline validation while providing stronger safety than naive memory or bandit-only methods.

2.4 Bandit-Based Hint Selection

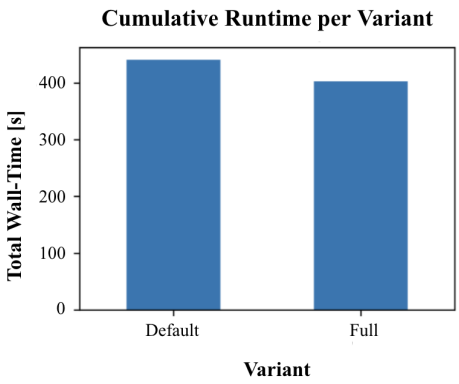
The final hint choice is made by a Thompson-sampling multi-armed bandit, which balances (i) the IMC's top prediction, (ii) any candidate from the EIM, and (iii) the database default. Each option is treated as an arm:

- A_{IMC}: the IMC-predicted best hint h_{IMC}, with posterior variance tied to $\sigma_{q,h}$.
- A_{EIM}: each distinct memory hint $h_{\text{mem},i}$, if available.
- A_{DFLT}: the default plan (no hint).

For arm a, the observed reward is defined as query speedup: $r_a = \ell_{\text{default}}/\ell_{q,a}$, where $\ell_{q,a}$ is the latency under hint a. Executions provide samples $\tilde{r}_a$ to update posteriors, and Thompson sampling selects the arm with the highest sampled reward. Including the default as an arm prevents slowdowns: if learned hints fail to beat it, the bandit converges to the default. When IMC or EIM arms consistently yield gains, the bandit favors them, enabling adaptive exploration while maintaining safety. This joint use of IMC, memory, and default provides stronger safeguards than memory-only or heuristic filters, which lack principled uncertainty handling.



(a) Tail-latency distribution across percentiles.



(b) Total latency by variant.

Figure 2: Default refers to vanilla PostgreSQL timings, while Full refers to DataSwift.

3 Preliminary Experiments

The preliminary experiments aim to address two questions:

1. Can DataSwift prevent all query regressions, specifically, major query regressions?
2. How effectively does DataSwift reduce tail latency and accelerate workloads?

Evaluation was conducted on PostgreSQL 16.1 [12] using the JOB and Extended JOB benchmarks [13, 14] over the IMDb database. Experiments ran on a Google Cloud VM with 16 vCPUs, 96 GB RAM, a 200 GB SSD, and Ubuntu 22.04 LTS. Vanilla PostgreSQL serves as the baseline.

3.1 Performance Regressions

A *major performance regression* is defined as any query at least 1.2× slower than vanilla PostgreSQL and adding 5+ seconds of latency (threshold chosen to ignore sub-second noise). A *catastrophic regression* is at least 3× slower with 30+ seconds extra runtime. Among 137 JOB queries, only one (0.7%) met the slowdown criteria, and none were catastrophic.

3.2 Latency Metrics

Performance is measured using median (50th) and tail (95th percentile) latencies. Figure 2a shows that DataSwift runs at $\sim 0.95\times$ the speed of PostgreSQL at the median, but achieves a **1.4× speedup** (41.5% faster) on the slowest 5% of queries. The slight median slowdown reflects conservative plan choices on simple queries, trading marginal losses for robustness, while tail queries benefit substantially. Aggregating runtime across the JOB workload (Fig. 2b), DataSwift yields an overall speedup of $\sim 1.1\times$. Unlike prior learned optimizers that rely on repeated queries, retraining, or heavy retrieval, DataSwift achieves end-to-end gains without such overhead, making it safer for deployment in latency-sensitive settings.

4 Conclusion and Future Work

This paper introduced DataSwift, an embedding-driven query optimizer combining inductive matrix completion, hybrid embeddings, and bandit-based selection. Preliminary results show reduced tail latency, overall speedups, and no catastrophic regressions. Some exciting future directions include:

Repetitive Queries. Offline exploration on recurring queries, as in LimeQO [5], could be combined with DataSwift's online adaptation to further cut total latency.

Cost-Aware Adapter Tuning. Lightweight adapters on the frozen Sentence-Transformers can be trained on SQL–latency pairs, shaping embeddings toward cost factors and improving efficiency and accuracy.

References

[1] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making Learned Query Optimization Practical. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*, pages 1275–1288, New York, NY, USA, 2021. Association for Computing Machinery. doi: 10.1145/3448016.3452838.

[2] Remmelt Ammerlaan, Gilbert Antonius, Marc Friedman, H. M. Sajjad Hossain, Alekh Jindal, Peter Orenberg, Hiren Patel, Shi Qiao, Vijay Ramani, Lucas Rosenblatt, Abhishek Roy, Irene Shaffer, Soundarajan Srinivasan, and Markus Weimer. Perfguard: Deploying ml-for-systems without performance regressions, almost! *Proceedings of the VLDB Endowment*, 14(13): 3362–3375, 2021. doi: 10.14778/3484224.3484233.

[3] Lianggui Weng, Rong Zhu, Di Wu, Bolin Ding, Bolong Zheng, and Jingren Zhou. Eraser: Eliminating Performance Regression on Learned Query Optimizer. *Proceedings of the VLDB Endowment*, 17(5):926–938, 2024. doi: 10.14778/3641204.3641205. URL `https://www.vldb.org/pvldb/vol17/p926-zhu.pdf`.

[4] Zixuan Yi, Yao Tian, Zachary G. Ives, and Ryan Marcus. Low Rank Approximation for Learned Query Optimization. In *Proceedings of the Seventh International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM '24)*, pages 1–6, New York, NY, USA, 2024. Association for Computing Machinery. doi: 10.1145/3663742.3663974. URL `https://doi.org/10.1145/3663742.3663974`.

[5] Zixuan Yi, Yao Tian, Zachary G. Ives, and Ryan Marcus. Low rank learning for offline query optimization. In *SIGMOD '25: 2025 ACM SIGMOD International Conference on Management of Data*, 2025. doi: 10.48550/arXiv.2504.06399. URL `https://doi.org/10.48550/arXiv.2504.06399`. To appear in SIGMOD 2025.

[6] Christoph Anneser, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, and Ryan Marcus. AutoSteer: Learned Query Optimization for Any SQL Database. *Proceedings of the VLDB Endowment*, 16(12):3515–3527, 2023. doi: 10.14778/3611540.3611544. URL `https://www.vldb.org/pvldb/vol16/p3515-anneser.pdf`.

[7] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. Lero: A Learning-to-Rank Query Optimizer. *Proceedings of the VLDB Endowment*, 16(6):1466–1479, 2023. doi: 10.14778/3583140.3583160. URL `https://www.vldb.org/pvldb/vol16/p1466-zhu.pdf`.

[8] Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. Roq: Robust Query Optimization Based on a Risk-aware Learned Cost Model. arXiv preprint arXiv:2401.15210, 2024.

[9] Peter Akioyamen, Zixuan Yi, and Ryan Marcus. The unreasonable effectiveness of llms for query optimization. In *Proceedings of the Machine Learning for Systems Workshop at NeurIPS 2024*, volume 6 of *Proceedings of Machine Learning and Systems*, pages 87–100, 2024. doi: 10.48550/arXiv.2411.02862. URL `https://doi.org/10.48550/arXiv.2411.02862`. To appear.

[10] Baoming Chang, Amin Kamali, and Verena Kantere. A novel technique for query plan representation based on graph neural nets. In *Proceedings of the 26th International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2024), Naples, Italy, August 26–28, 2024*, volume 14912 of *Lecture Notes in Computer Science*, pages 299–314, Cham, Switzerland, August 2024. Springer. doi: 10.1007/978-3-031-68323-7_25. URL `https://doi.org/10.1007/978-3-031-68323-7_25`.

[11] Nagarajan Natarajan and Inderjit S. Dhillon. Inductive matrix completion for predicting gene–disease associations. *Bioinformatics*, 30(12):i60–i68, June 2014. doi: 10.1093/bioinformatics/btu269. URL `https://doi.org/10.1093/bioinformatics/btu269`. Open Access.

[12] Postgresql database. http://www.postgresql.org/, 2025. URL `http://www.postgresql.org/`.

[13] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9 (3):204–215, 2015. doi: 10.14778/2850583.2850594. URL `https://doi.org/10.14778/2850583.2850594`.

[14] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A Learned Query Optimizer. *Proceedings of the VLDB Endowment*, 12(11):1705–1718, 2019. doi: 10.14778/3342263.3342644. URL `https://www.vldb.org/pvldb/vol12/p1705-marcus.pdf`.