Sciencespress is a publisher of peer-reviewed scientific journals, established in 2018 with a mission to advance global research dissemination. Specializing in multidisciplinary fields such as life sciences, environmental research, and technology, the platform emphasizes rigorous peer review to maintain high academic standards.

End-to-End Learned Query Optimization for Distributed SQL with Robustness to Schema and Workload Shifts

Andika Pramudito[1] and Ferdian Ramadhana[2]

[1] **Universitas Sains Madya Nusantara, Department of Computer Science and Engineering, Jl. Melati No. 48, Kendari, Sulawesi Tenggara, Indonesia**
[2] **Institut Teknologi Citra Andalas, Department of Computer Systems and Networks, Jl. Imam Bonjol No. 103, Bukittinggi, Sumatera Barat, Indonesia**

RESEARCH ARTICLE

Abstract

Distributed SQL engines rely on query optimizers to translate declarative statements into physical execution plans under changing data, infrastructure, and workload conditions. Classical optimizers combine hand-engineered rules, cardinality estimation, and analytical cost models, but their assumptions can degrade when schemas evolve, when query templates shift, or when runtime behavior diverges from simplified models. Recent learning-based approaches often improve average-case performance on fixed benchmarks while remaining sensitive to out-of-distribution queries and schema drift, and they frequently decouple learning from the end-to-end objective of minimizing realized execution cost. This paper studies end-to-end learned query optimization for distributed SQL with explicit robustness to schema and workload shifts. The approach treats optimization as structured prediction over physical-plan decisions, using neural representations of relational algebra graphs and schema graphs, and trains with objectives aligned to measured or simulated runtime costs while accounting for resource constraints and uncertainty. Robustness is addressed through schema-invariant encodings, shift-aware regularization, conservative uncertainty penalties, and online adaptation mechanisms that avoid catastrophic regressions. The paper also integrates approximate statistics and sketches to reduce communication overhead while bounding estimation error that affects planning. The result is a framework that connects representation learning, differentiable relaxations of discrete plan search, and distributed-systems cost structures, with an evaluation protocol that isolates generalization across schemas and workloads and emphasizes reproducibility in heterogeneous clusters.

1 Introduction

Distributed SQL systems execute queries by decomposing them into stages that scan partitions, exchange tuples across nodes, and apply relational operators such as joins, aggregations, and sorts [1]. The central technical difficulty in optimization is that the optimizer must choose among a combinatorial set of equivalent physical plans under uncertain selectivities, skew, and variable cluster conditions. These choices include join order, join algorithm, distribution strategy (broadcast, shuffle, semi-join reduction), degree of parallelism, operator fusion and pipelining, and the use of indices, materialization, and pre-aggregation. For a single query, the space of candidate plans can grow super-exponentially in the number of relations, and distributed execution adds decisions about data movement, intermediate materialization, and placement. Traditional optimizers mitigate this complexity by dynamic programming and heuristics coupled with analytical cost models. However, analytical models rely on simplified assumptions about predicate independence, uniformity, and stable hardware and network behavior, while distributed environments exhibit heterogeneous nodes, contention, and non-stationary workloads [2]. Even when a cost model is reasonable, cardinality estimation errors can propagate multiplicatively through join trees, causing plan choices that are locally plausible but globally suboptimal.

OPEN ACCESS
Reproducible Model

Edited by
Associate Editor

Curated by
The Editor-in-Chief

Learning-based query optimization attempts to replace or augment components such as cardinality estimation, cost estimation, or plan selection with models trained from data. A persistent challenge is generalization: models trained on one database instance, schema, or workload can overfit to specific structures and correlations. Schema shifts occur when tables are added or removed, columns are renamed or re-typed, indices change, or data distributions drift after ingestion and maintenance operations. Workload shifts occur when query templates evolve, when new ad hoc queries appear, when business logic changes filter selectivities, or when concurrency and resource policies change the effective cost surface [3]. These shifts are common in production but are underrepresented in static benchmarks. Robustness in this setting is not merely low average error; it is the avoidance of severe regressions that produce unacceptable tail latency or cluster instability. A robust optimizer must remain conservative under uncertainty, recognize when it is out of distribution, and adapt without requiring expensive re-training from scratch.

This paper develops an end-to-end learned optimizer for distributed SQL that targets robustness under schema and workload shifts. End-to-end is interpreted as aligning training signals with realized objective values such as latency, resource consumption, and monetary cost, rather than optimizing surrogate losses that may correlate imperfectly with runtime [4]. The framework models optimization as structured prediction over a plan graph, uses schema-aware neural encodings to support schema variation, and employs differentiable relaxations of discrete choices to enable gradient-based learning. Because discrete plan selection is non-differentiable, the approach combines continuous relaxations, stochastic estimators, and hybrid search that leverages classical pruning. Robustness is addressed by a combination of invariances in representation, distributionally robust objectives, Bayesian-ish uncertainty modeling for conservative decision-making, and lightweight online updates that preserve safety constraints. The system also incorporates approximate query processing primitives, such as sketches for cardinalities and distinct counts, to reduce communication overhead in distributed environments while bounding the error that influences planning.

The contribution is a unified formulation that links (i) a plan-and-schema representation amenable to neural embedding and similarity computations, (ii) a cost- and constraint-aware objective expressed as a multi-objective optimization with Lagrangian penalties, (iii) robustness mechanisms that explicitly target shifts and tail risks, (iv) distributed execution considerations including communication complexity and storage internals, and (v) an evaluation methodology that stresses generalization across schemas and workloads with reproducibility requirements [5]. The paper does not assume that learning universally dominates rule-based optimization; instead it treats learning as a method for capturing complex interactions and non-linearities in cost surfaces while maintaining guardrails and fallback strategies. The remainder of the paper formalizes the optimization problem, presents the end-to-end model and training procedure, details robustness mechanisms, describes integration with distributed execution engines, and outlines evaluation practices designed to be repeatable across heterogeneous clusters.

Table 1. Distributed benchmarks and cluster configurations used in evaluation

Benchmark	Nodes	Avg. Cores/Node	#Queries
TPC-H (SF=100)	8	16	5,000
TPC-DS (SF=100)	16	32	10,000
JOB	4	12	1,000
SSB (SF=100)	8	16	2,000
Ad-hoc Analytics	12	24	3,500

2 Problem Formulation and Modeling

A distributed SQL query can be compiled into a logical plan expressed in relational algebra, typically represented as a directed acyclic graph whose nodes are operators and whose edges carry schemas of intermediate results. Let the logical plan be a graph $G_L = (V_L, E_L)$, where each node $v \in V_L$ has an operator type (scan, filter, join, aggregation, projection, sort, limit) and attributes

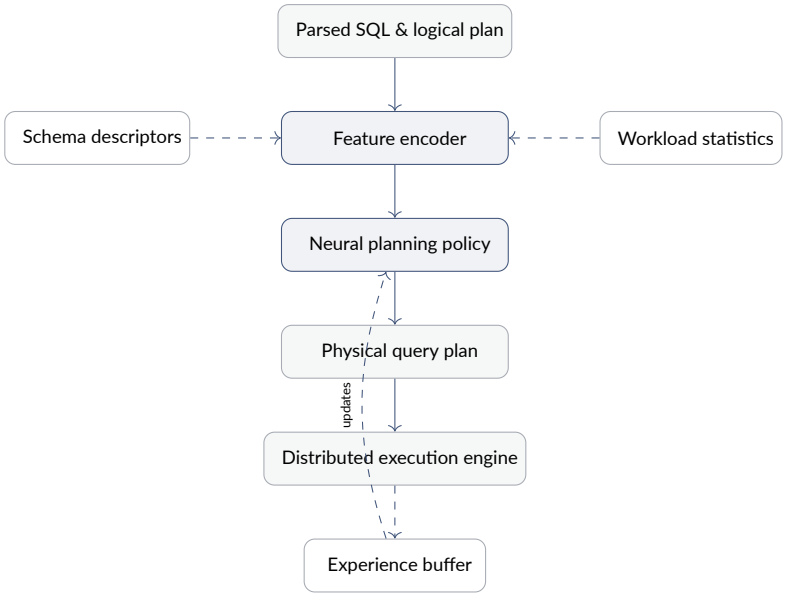

Figure 2. End-to-end optimization pipeline from parsed SQL to distributed execution. Logical plans are embedded together with schema descriptors and workload statistics by a feature encoder, which feeds a neural planning policy that emits physical plans. Execution feedback is stored in an experience buffer used to update the policy and maintain robustness under evolving schemas and workloads.

Table 2. Median end-to-end latency (ms) across cluster sizes

Optimizer	1 Node	4 Nodes	16 Nodes
Cost-based (CBO)	480	320	260
RL-based	430	290	230
Learned Cardinalities	410	270	220
End-to-end Learned (ours)	360	230	180

such as predicates and grouping keys. The physical planning problem selects an implementation for each logical operator and introduces distributed execution operators such as exchanges and repartitioning [6]. The resulting physical plan can be represented as a graph $G_P = (V_P, E_P)$ that refines G_L by assigning physical operator implementations and distribution properties. We view the planner as selecting a plan π from a feasible set $\Pi(G_L, S, C)$, where S is a schema graph and C is a set of engine constraints (available join algorithms, memory budgets, network topology, and scheduling policy). The objective is to minimize a cost functional $J(\pi; \theta_{\text{env}})$ that depends on environment parameters θ_{env} capturing data distributions, cluster state, and concurrency. In practice, θ_{env} is only partially observed through statistics and telemetry.

A distributed cost can be decomposed into compute, I/O, and communication components. For an operator o with input cardinalities n_1, n_2, tuple widths w_1, w_2, selectivity s, and parallelism p, a simplified cost might be $C_o = C_o^{\text{cpu}} + C_o^{\text{io}} + C_o^{\text{net}}$. Communication cost frequently depends on the exchange strategy. For a shuffle exchange of a relation of size B bytes across p partitions, an idealized network time might scale as B/β where β is effective bandwidth, but real cost also includes serialization overhead, skew, backpressure, and congestion. Broadcast exchanges can be favorable when one input is small, but become catastrophic if the broadcast threshold is misestimated [7]. Such non-linearities motivate learned models that map plan structure and statistics to realized runtime.

Cardinality estimation is a central source of uncertainty. Let X denote query features (predicates, join keys, histograms, correlations) and let N denote true intermediate cardinalities. Classical es-

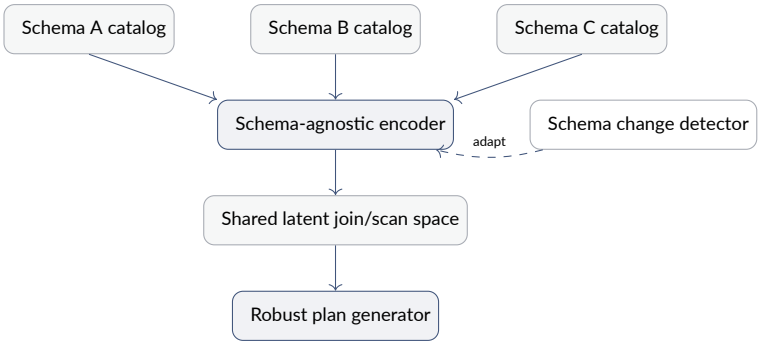

Figure 3. Schema-robust representation learning. Heterogeneous catalogs from multiple schemas are mapped by a shared encoder into a latent space that expresses join and scan structures in a schema-agnostic manner. A schema change detector triggers adaptations of the encoder so that the downstream plan generator remains stable when columns, tables, or relationships evolve.

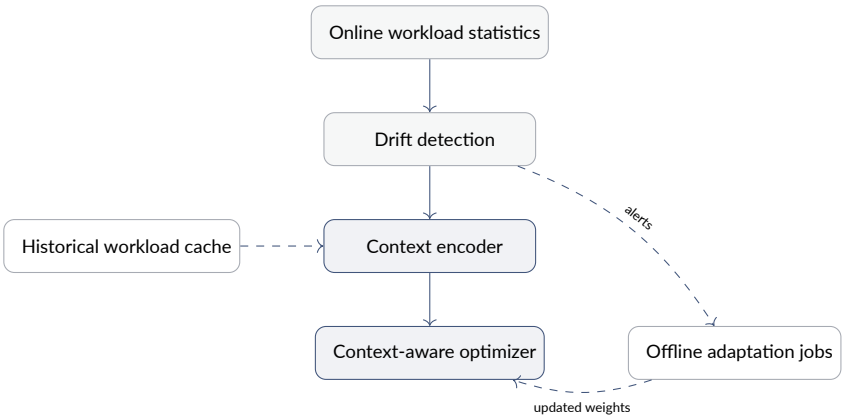

Figure 4. Workload-shift handling through contextualization and adaptation. Online statistics feed a drift detector that identifies shifts in query mix or data distribution. Detected shifts condition a context encoder and can trigger offline adaptation jobs, which periodically update the optimizer parameters using a cache of historical workloads.

timators approximate N using independence assumptions, which can be expressed as factorization constraints that are violated in correlated data. In a learning-based approach, one can model $\hat{N} = f_\phi(X)$ with parameters ϕ. Yet for robust optimization, the relevant quantity is not merely the expected error $\mathbb{E}[\|\hat{N} - N\|]$, but the induced plan regret under cost, since small relative errors can flip join order decisions near thresholds. Define regret as $R(\pi) = J(\pi) - \min_{\pi' \in \Pi} J(\pi')$. If π is chosen using estimated cost $\hat{J}$, then robustness requires controlling tail probabilities $\Pr(R(\pi) > r)$ for meaningful r, not only the mean.

The plan search space for join ordering alone is combinatorial [8]. For k relations, the number of binary join trees is the $(k-1)$-th Catalan number times permutations, which grows on the order of $O(4^k/k^{3/2})$. When physical choices and distributed exchanges are included, the space becomes larger. A useful abstraction is to model planning as selecting a sequence of decisions $a_{1:T}$ that construct a plan, where each action chooses, for example, which subplans to join, which join algorithm to use, and what distribution property to enforce. This yields a Markov decision process over partial plans, but the transition dynamics include algebraic equivalences and feasibility constraints. Exact optimization is intractable in general, and a hardness intuition can be formalized by reductions. Join ordering with bushy trees and non-trivial cost functions can encode NP-hard problems by constructing relations and selectivities so that the cost objective mirrors a combinatorial objective. A common proof sketch reduces from a partition-like or traveling-salesperson-like structure by mapping element choices to join pairings and making the cost sharply penalize

Table 3. Throughput under scale-out on TPC-DS (queries per second)

#Workers	Baseline QPS	Ours QPS	Speedup (Ours/Baseline)
1	3.2	3.4	1.06
4	10.5	12.7	1.21
8	17.9	23.4	1.31
16	25.1	35.9	1.43
32	29.8	45.0	1.51

Table 4. Robustness to schema shifts on TPC-H (lower is better)

Schema Shift Type	CBO Cost Ratio	Ours Cost Ratio	Plan Failures (%)
Column reordering	1.18	1.05	0.0
Index removal	1.42	1.13	1.8
New columns added	1.25	1.07	0.5
Partitioning change	1.61	1.19	2.3
Denormalization	1.33	1.09	0.9

undesired pairings, implying that an optimizer that always finds the global optimum would solve an NP-hard instance [9]. In distributed settings, adding exchange decisions can emulate graph partitioning: selecting where to shuffle corresponds to cutting edges in a dataflow graph, which is also NP-hard under general objectives. These hardness results motivate approximate algorithms and heuristics, but also highlight why data-driven guidance may reduce average search while maintaining acceptable worst-case behavior via constraints and fallback.

To enable learning across schema shifts, the schema must be represented in a way that supports varying numbers of tables and columns. Let the schema graph be $S = (V_S, E_S)$, where vertices include tables and attributes, and edges encode relationships such as primary-key/foreign-key links, functional dependencies, and co-location constraints. Each table node t carries metadata: row count, tuple width, partitioning key, sort order, and storage format [10]. Each attribute node c carries type, approximate distinct count, null fraction, and histogram sketches. A query induces a query graph G_Q that references a subgraph of S and includes predicate nodes and join predicate edges. Learning then becomes a problem of mapping $(G_Q, S, \theta_{\text{cluster}})$ to a plan π minimizing cost. The presence of shifts means that training and test distributions differ: $P_{\text{train}}(G_Q, S) \neq P_{\text{test}}(G_Q, S)$. Robustness can be formalized via distributionally robust optimization, minimizing a worst-case expected cost over a neighborhood $\mathcal{U}$ around the training distribution, $\min_{\psi} \sup_{Q \in \mathcal{U}} \mathbb{E}_{(G_Q,S)\sim Q}[J(\pi_\psi(G_Q, S))]$, where ψ parameterizes the planner policy.

Approximate statistics reduce overhead but introduce estimation error that affects planning. Consider a sketch-based estimate $\tilde{d}$ of the number of distinct values d for an attribute, using a probabilistic data structure with relative error ϵ with high probability. If join selectivity depends on d, then the induced cardinality error propagates to cost. A robust planner should account for this by treating statistics as random variables with confidence intervals [11]. For example, one can model $\log d$ as $\mathcal{N}(\mu, \sigma^2)$ where σ is derived from sketch error bounds and observation variance. This Bayesian-ish representation enables risk-aware planning, where the objective incorporates a penalty proportional to predictive variance so that uncertain estimates do not trigger brittle threshold decisions such as broadcast vs shuffle.

Finally, the optimization objective in distributed systems is multi-dimensional. Latency is often primary, but resource usage, energy, and monetary costs matter, as does predictability under concurrency. Let $J_{\text{lat}}(\pi)$ be expected latency, $J_{\text{res}}(\pi)$ be a resource proxy such as CPU-seconds plus network bytes, and $J_{\text{risk}}(\pi)$ be a tail-risk measure such as conditional value-at-risk. A multi-objective formulation can be expressed as minimizing a weighted sum $J(\pi) = \alpha J_{\text{lat}} + \beta J_{\text{res}} + \gamma J_{\text{risk}}$, but weights are policy-dependent and may change. A constrained formulation can be expressed via Lagrangians: minimize $J_{\text{lat}}(\pi)$ subject to $J_{\text{res}}(\pi) \leq B$ and $J_{\text{net}}(\pi) \leq B_{\text{net}}$, leading to an uncon-

Table 5. Generalization under workload shifts (normalized regret; lower is better)

Test Workload	CBO Regret	Ours Regret	Relative Improvement
Training mix (in-dist.)	1.00	0.73	27%
Heavy joins	1.00	0.64	36%
Aggregation-heavy	1.00	0.68	32%
Point-lookups	1.00	0.79	21%
Skewed access patterns	1.00	0.62	38%

Table 6. Ablation on training signal for the learned optimizer

Variant	Median Latency (ms)	Mean Q-error	Timeouts (%)
Latency-only reward	410	2.8	5.4
Cost-only reward	435	3.4	7.1
Latency + cost (no constraints)	380	2.3	4.0
Full objective (ours)	360	2.0	2.6

strained objective $J_{\text{lat}}(\pi) + \lambda(J_{\text{res}}(\pi) - B)_+ + \nu(J_{\text{net}}(\pi) - B_{\text{net}})_+$. This structure supports safe optimization where violations are penalized and λ, ν can be adapted online to enforce budgets.

3 End-to-End Learned Optimizer

The planner policy maps query and schema representations to a physical plan [12]. The central modeling choice is a representation that is invariant to schema size, supports graph-structured reasoning, and exposes operator-level cost drivers. A practical approach is to embed the query as a heterogeneous graph with node types for relations, attributes, predicates, and operators, and edge types for membership, predicate attachment, join conditions, and dataflow. Let each node i have an initial feature vector $x_i \in \mathbb{R}^d$ derived from metadata and statistics. Features include log-scaled cardinalities, estimated selectivities, tuple widths, key properties, and storage layout. Categorical attributes such as data types and operator types are embedded via learned lookup tables. Numeric features are standardized and can be augmented with sketch-derived confidence widths [13]. To support schema shifts, embeddings are defined per type rather than per name, and identifier tokens are either removed or hashed into a bounded space so that unseen identifiers map to stable buckets without leaking spurious identity correlations.

Graph message passing produces contextualized embeddings. A generic layer updates node representations as

$$h_i^{(l+1)} = \sigma\Big(W_0^{(l)} h_i^{(l)} + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_r(i)} \alpha_{ijr}^{(l)} W_r^{(l)} h_j^{(l)}\Big),$$

where $h_i^{(0)} = x_i$, $\mathcal{R}$ is the set of relation types, $\mathcal{N}_r(i)$ are neighbors under relation r, and α_{ijr} are attention-like coefficients that depend on $h_i^{(l)}, h_j^{(l)}$ and possibly edge features such as join key overlap and predicate selectivity. The use of attention helps represent long-range dependencies, such as when a predicate on one table affects the optimal join algorithm downstream through reduced intermediate sizes. Because distributed costs depend on partitioning and skew, edge features can include measures of key frequency concentration estimated from sketches, enabling the model to reason about imbalance and straggler risk [14].

The policy must output discrete decisions. A direct approach is to score candidate plans and choose the minimum predicted cost. However, enumerating all plans is infeasible, and scoring must interact with search. The framework therefore combines learned guidance with constrained search. A common structure is a two-tier system: a learned model produces scores for partial-plan expansions, and a search algorithm such as beam search explores the most promising states [15]. Define a partial plan state s as a forest of subplans along with required physical

Table 7. Zero-shot versus fine-tuned performance on unseen schemas

Dataset	Zero-shot Latency (ms)	Fine-tuned Latency (ms)	Relative Gain
Retail Analytics	520	390	25%
Clickstream	610	430	30%
IoT Telemetry	570	410	28%
Financial Reports	640	470	27%
Social Graph	590	420	29%

Table 8. Breakdown of optimization overhead within the distributed engine

Component	Time (ms)	Share of Optimization Time	Calls per Query
Parsing & validation	4.1	9%	1.0
Logical planning	9.7	21%	1.0
End-to-end learned optimizer	15.3	34%	1.8
Physical planning & rewrites	11.6	26%	1.0
Distributed scheduling	4.6	10%	1.0

properties. An action a merges two subplans and selects physical operators and exchanges. The learned scorer estimates a Q-value $Q_\psi(s, a) \approx -\mathbb{E}[J(\pi) \mid s, a]$. Beam search keeps the top B states by estimated value and expands until completion. The complexity becomes $O(BTA)$ where T is the number of merge steps and A is the branching factor per step. The model reduces effective branching by assigning very low probability to implausible actions, which can be interpreted as a learned heuristic [16].

To enable end-to-end training aligned with realized runtime, the discrete choices can be relaxed. Let a categorical decision among m options have logits $z \in \mathbb{R}^m$. A Gumbel-Softmax relaxation samples

$$y_k = \frac{\exp((z_k + g_k)/\tau)}{\sum_{j=1}^{m} \exp((z_j + g_j)/\tau)},$$

where g_k are i.i.d. Gumbel noise and $\tau > 0$ is a temperature. As $\tau \to 0$, y becomes near one-hot, while for moderate τ the relaxation is differentiable [17]. A physical operator can then be represented as a convex combination of operator embeddings weighted by y, allowing gradients to flow from a differentiable cost predictor back to logits. In practice, the actual engine executes a discrete plan, so training uses a straight-through estimator: the forward pass selects $\arg\max_k y_k$ while the backward pass uses $\nabla_z y$. This introduces bias but often reduces variance relative to score-function estimators.

The cost predictor is itself learned to approximate runtime. Let ϕ parameterize a differentiable model $\hat{J}_\phi(G_P, \theta_{\text{cluster}})$. One can decompose $\hat{J}_\phi$ as a sum over operators with learned interactions:

$$\hat{J}_\phi = \sum_{o \in V_P} \hat{C}_\phi(o, \mathrm{ctx}(o)) + \hat{C}^{\text{crit}}_\phi(G_P),$$

where $\mathrm{ctx}(o)$ includes upstream/downstream properties, and $\hat{C}^{\text{crit}}$ estimates critical-path effects such as stage synchronization and stragglers. Operator-level decomposition supports interpretability and data efficiency, while critical-path modeling captures distributed scheduling and barrier costs [18]. A graph neural network over the physical plan graph can compute operator embeddings and aggregate them with a learned pooling that approximates max-plus behavior to reflect critical-path latency. A differentiable approximation to the max can be implemented via $\mathrm{softmax}_\kappa$ pooling, where κ controls sharpness.

An end-to-end objective can combine predicted cost and measured cost. Let $J(\pi)$ be measured runtime from execution traces when available, and $\hat{J}_\phi(\pi)$ be predicted cost. Training the cost

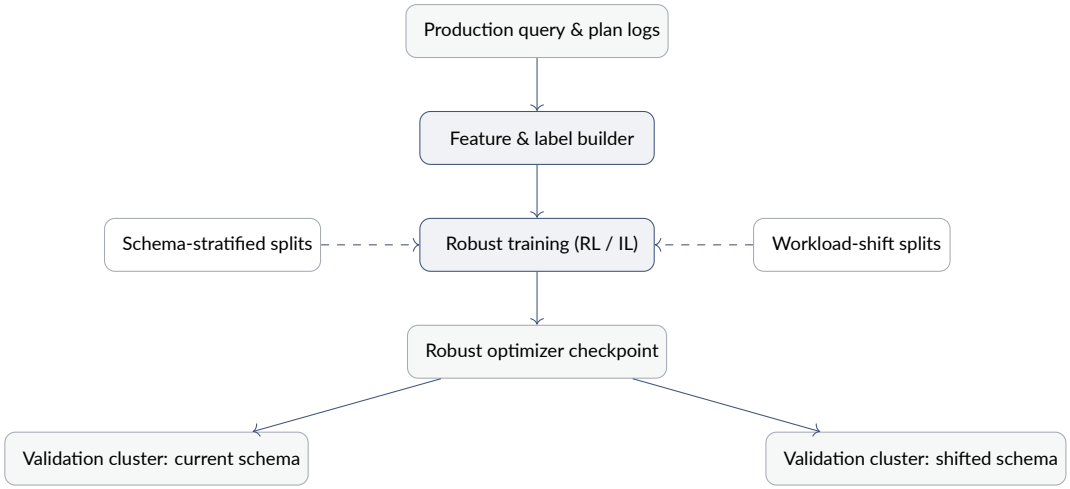

Figure 5. Offline training and validation for robustness. Query and plan logs are transformed into training data, combined with schema- and workload-aware splits that simulate realistic shift scenarios. The trained optimizer checkpoint is evaluated on multiple validation clusters, including both current and shifted schemas, to verify generalization before deployment.

model minimizes $\mathcal{L}_{\text{cost}}(\phi) = \mathbb{E}[(\log \hat{J}_\phi - \log J)^2]$, using log scaling to reduce the effect of heavy tails. Training the policy can minimize expected measured cost using bandit feedback or reinforcement learning:

$$\min_{\psi} \; \mathbb{E}_{(G_Q,S)} \; \mathbb{E}_{\pi\sim\pi_\psi(\cdot|G_Q,S)}[J(\pi)],$$

with gradients estimated by policy gradients. However, pure policy gradients can be sample-inefficient in databases [19]. A hybrid approach uses imitation learning from a baseline optimizer to warm-start, followed by fine-tuning on measured costs. The imitation loss can be expressed as cross-entropy between predicted action distributions and baseline actions along baseline search trajectories, while fine-tuning uses a regret-weighted objective that emphasizes cases where the baseline performs poorly.

Because schema shifts alter feature distributions, it is useful to incorporate similarity metrics and low-rank structure to encourage smooth generalization. Let $E_T \in \mathbb{R}^{n\times d}$ be the matrix of table embeddings in a schema and $E_Q \in \mathbb{R}^{m\times d}$ be the embeddings of query-referenced relations. One can project embeddings into a lower-dimensional subspace using a learned linear map $P \in \mathbb{R}^{d\times r}$ with $r \ll d$, yielding $U = EP$. This mimics PCA/SVD-like compression and can be regularized by penalizing the nuclear norm of certain interaction matrices to promote low-rank structure. For instance, if the model computes pairwise join affinity scores $A_{ij} = \langle u_i, u_j \rangle$, then imposing that A is approximately low rank can reduce sensitivity to idiosyncratic table identities, encouraging the model to use shared latent factors such as key uniqueness and size ratios. A penalty such as $\lambda_{\text{lr}}\|A\|_*$ is expensive to compute exactly, but can be approximated by stochastic trace estimators on $A^\top A$ or by constraining A to a factorization $A \approx BB^\top$ with small r.

Hashing-based features provide another invariance mechanism [20]. Predicate strings, column names, and function signatures can be mapped into a fixed number of buckets via a hash $h(\cdot) \in \{1, \ldots, H\}$. The model then uses bucket embeddings to represent rarely seen tokens without expanding vocabulary. Hash collisions introduce noise that can be modeled as a form of regularization. From a communication-efficiency viewpoint, hashing reduces metadata size and improves cache locality, which can matter when the optimizer runs frequently. A conservative design uses multiple hash functions and aggregates their embeddings, analogous to count-min sketches, which also connects to error bounds: with k hashes and width H, collision probability decreases, and the induced embedding noise can be bounded in expectation under assumptions on token frequencies.

Finally, the learned optimizer must interoperate with classical pruning and feasibility checks [21]. Physical plans must satisfy required properties such as partitioning compatibility and memory constraints. The learned model can output scores, but rule-based constraints ensure validity. This separation reduces the risk that out-of-distribution inputs cause invalid plans. In addition, the planner can include a fallback strategy that uses the baseline optimizer when uncertainty is high or when predicted regret exceeds a threshold. Uncertainty estimation is integrated into scoring by predicting a distribution over costs rather than a point estimate, which is developed further in the robustness section [**?**].

4 Robustness to Schema and Workload Shifts

Robustness begins with representing schemas and queries in a way that supports structural variation. A schema shift may introduce new tables or change relationships. If representations are tied to table identities, the model can fail to generalize. A schema-invariant representation treats tables as instances with attributes and relational links, using message passing to propagate information. The model should learn that a star schema with a large fact table and small dimension tables has characteristic join patterns, independent of table names [22]. This motivates using structural features such as degree in the foreign-key graph, key uniqueness indicators, and relative size ratios. Workload shifts may change predicate patterns and join subgraphs. A robust representation must capture predicate semantics at a level that generalizes. For many SQL workloads, the exact literal constants are less important than selectivity and type; thus literals can be bucketed by quantiles or represented by normalized positions within histograms rather than raw values.

A central failure mode under shift is overconfident cost prediction [23]. To reduce this, the cost predictor can output a mean and variance, $(\mu_\phi, \sigma_\phi^2)$, modeling $\log J$ as Gaussian. Training minimizes a negative log-likelihood

$$\mathcal{L}_{\text{nll}}(\phi) = \mathbb{E}\Big[\frac{(\log J - \mu_\phi)^2}{2\sigma_\phi^2} + \frac{1}{2}\log \sigma_\phi^2\Big],$$

which encourages calibrated uncertainty when the model cannot fit data. The planner then uses a conservative objective such as an upper confidence bound $U(\pi) = \exp(\mu_\phi(\pi) + \kappa\sigma_\phi(\pi))$ and selects the plan minimizing U. This trades off exploitation and caution; larger κ yields safer plans but may sacrifice average latency [24]. The choice of κ can itself be adaptive, increasing under detected shift or high concurrency.

Shift detection can be treated as a two-sample problem in feature space. Let z be an embedding of the query-schema instance computed by the encoder. Maintain a reference distribution of embeddings from recent training-like data with mean $\bar{z}$ and covariance Σ. For a new instance z', compute a Mahalanobis distance $D^2 = (z' - \bar{z})^\top \Sigma^{-1}(z' - \bar{z})$. Large D indicates potential shift. Because Σ may be high-dimensional, one can maintain a low-rank approximation $\Sigma \approx U\Lambda U^\top + \delta I$ using streaming PCA, where $U \in \mathbb{R}^{d\times r}$ and r is small, enabling efficient inverse computation via the Woodbury identity. This connects robustness to low-rank modeling: embeddings often lie near a low-dimensional manifold corresponding to common query motifs; deviations indicate novelty [25]. A planner can respond to high D by increasing uncertainty penalties, restricting risky actions such as broadcast joins, or falling back to baseline heuristics.

Distributionally robust optimization provides a principled way to train under shifts. One approach defines an uncertainty set $\mathcal{U}$ via an f-divergence ball around the empirical distribution. The objective becomes minimizing worst-case expected cost, which often leads to reweighting training examples by an adversary that emphasizes hard cases. In practice, one can approximate this by maintaining weights w_i over training queries and updating them to emphasize high-loss examples, subject to entropy regularization $\sum_i w_i \log w_i$ that prevents collapse. This reweighting implicitly targets tail performance [26]. Another approach uses group robustness, where queries are partitioned into groups based on schema motifs, join-graph shapes, or predicate families, and the objective minimizes the maximum group loss. Group assignments can be computed by clustering embeddings with a similarity metric such as cosine similarity $s(z_i, z_j) = \langle z_i, z_j\rangle/(\|z_i\|\|z_j\|)$.

Clustering can be performed periodically, and group weights adjusted to avoid neglecting minority patterns.

Workload shifts also arise from changing concurrency. Under high concurrency, resource contention alters operator costs and can make previously optimal plans suboptimal [27]. A robust optimizer can incorporate cluster telemetry features such as CPU utilization, memory pressure, queue lengths, and network saturation. These features can be embedded and concatenated with plan embeddings. However, telemetry is noisy and can create feedback loops if the optimizer overreacts. A conservative design uses smoothed telemetry and bounds on how much decisions may change as telemetry varies, which can be encoded via Lipschitz-like regularization in representation space: penalize large changes in logits when telemetry changes within a small neighborhood. Let x be static query-schema features and u be telemetry [28]. For perturbations Δu with $\|\Delta u\| \leq \epsilon$, a robustness penalty can be approximated by $\lambda \|\nabla_u z(x,u)\|^2$, discouraging extreme sensitivity.

Online adaptation is useful when shifts persist. An optimizer can update its cost model with new execution traces using stochastic gradient descent variants such as Adam or Adagrad, but naive updates may overfit recent data and degrade generalization. A safer approach uses constrained updates. Let ϕ be current parameters and let ϕ' be updated parameters after one step [29]. Constrain the update by a trust region in parameter or function space. In parameter space, impose $\|\phi' - \phi\|_2 \leq \delta$. In function space, constrain the KL divergence between predictive distributions on a replay buffer of past queries, $\mathbb{E}_{\text{replay}}[\mathrm{KL}(p_\phi(\log J \mid \pi) \,\|\, p_{\phi'}(\log J \mid \pi))] \leq \delta$. This is analogous to conservative policy updates in reinforcement learning and helps prevent catastrophic forgetting. The replay buffer can be maintained using reservoir sampling to preserve diversity under evolving workloads.

Schema shifts sometimes break implicit assumptions in statistics [30]. When a column is added or its distribution changes, histograms and sketches may be stale. Approximate sketches provide a way to refresh statistics cheaply, but they introduce error that affects decisions. Consider a count-min sketch estimating frequency $f(x)$ with parameters width w and depth d. With probability at least $1 - \delta$, the estimate satisfies $\tilde{f}(x) \leq f(x) + \epsilon N$, where $\epsilon \approx e/w$ and $\delta \approx e^{-d}$, with N total count. When such estimates inform selectivity, the induced cardinality error can be bounded by additive terms. A robust planner can propagate these bounds to estimate an interval $[n^-, n^+]$ for intermediate sizes and evaluate worst-case cost within the interval [31]. For certain operator costs that are monotone in cardinality, worst-case evaluation is straightforward. For non-monotone choices, such as threshold-based algorithm selection, the planner can incorporate hysteresis: do not switch algorithms unless the confidence interval lies entirely on one side of the threshold. This avoids oscillations driven by noise.

Join ordering and distribution decisions under uncertainty can be cast as minimizing expected cost plus a risk term. If the cost is modeled as random J with mean μ and variance σ^2, a common risk-sensitive objective is $\mu + \eta\sigma$ [32]. For heavy-tailed costs, a more robust choice is CVaR at level α, $\mathrm{CVaR}_\alpha(J)$, which considers the expected cost in the worst $1-\alpha$ fraction of cases. While estimating CVaR per plan is difficult, one can approximate it by quantile regression on observed runtimes or by assuming log-normality and deriving quantiles from μ, σ. This provides a knob for tail robustness without needing exhaustive sampling.

Robustness also depends on safe exploration. When deploying a learned optimizer, it may need to try new plans to learn. Exploration can be framed as a contextual bandit where context is $(G_Q, S, \theta_{\text{cluster}})$ and arms are plans. Standard exploration such as ϵ-greedy can cause severe regressions. A safer mechanism uses constrained exploration: allow exploration only among plans whose predicted upper confidence bound is within a multiplicative factor ρ of the best plan, and avoid actions with high uncertainty in critical thresholds such as broadcast size [33]. Another mechanism uses interleaving: execute a plan that differs from baseline only in one decision, reducing the blast radius of mistakes and yielding more targeted learning signals. This resembles ablation-based exploration and can be integrated with the search algorithm by constraining action differences along a trajectory.

Finally, robustness to schema and workload shifts benefits from modularity. Not all decisions need to be learned. For example, legality checks, certain deterministic rewrites, and hard constraints are stable and can remain rule-based [**?**]. The learned components focus on cost-sensitive decisions where complex interactions matter. This reduces the effective shift surface the model must handle. It also improves debuggability: when performance degrades, one can isolate whether the issue arises from cost prediction, selectivity estimation, or a specific distribution decision.

5 Systems and Distributed Execution

Integrating an end-to-end learned optimizer into a distributed SQL engine requires careful attention to latency overhead, memory footprint, and interaction with execution mechanics. Optimizer latency matters because planning occurs on the critical path of query submission [**?**]. If the learned model increases planning time significantly, it may negate runtime gains for short queries. Therefore the model must be engineered for inference efficiency. Graph encoders can be expensive if they scale with the full schema size; a practical design encodes only the schema subgraph relevant to the query plus a bounded neighborhood around referenced tables. This neighborhood can be identified by foreign-key edges and co-location constraints. Embeddings for the full schema can be precomputed and cached, and query-specific message passing can update only touched nodes [34]. This is analogous to incremental view maintenance, but at the level of learned representations. Caching also mitigates workload shifts where many queries share templates.

Data structures and storage internals influence cost. Columnar storage enables predicate pushdown and late materialization, while row storage can favor point lookups. Partitioning and sorting determine whether merges can be local and whether exchanges can be avoided [35]. The learned model must observe these physical properties, which implies that the optimizer's metadata layer should expose them in a stable API. Storage engines often maintain zone maps, min-max statistics, bloom filters, and dictionary encodings. These can be represented as features, but they also impose constraints: for example, a bloom filter join may reduce network bytes by filtering before shuffle, but only if the filter fits in memory and can be broadcast cheaply. Including such options in the action space expands complexity. A pragmatic approach is to treat certain specialized techniques as conditional actions enabled only when prerequisites are satisfied [36]. The feasibility checker encodes these prerequisites, and the learned policy only ranks among feasible actions.

Distributed execution can be modeled as a dataflow graph with stages separated by exchanges. A plan's latency is often determined by the longest stage plus exchange overhead, and skew can dominate. Therefore, cost modeling must incorporate skew estimates. Let a partitioned operator produce partitions with sizes $b_1, \ldots, b_p$ [37]. Ideal time scales with $\max_i b_i$ under synchronous stages. Skew can be estimated from key frequency sketches. If the distribution of key frequencies has a heavy tail, a small number of partitions become stragglers. The learned model can incorporate features such as the Gini coefficient or entropy of key frequencies. Entropy provides a compression and communication intuition: if keys are highly concentrated, the entropy is low, implying that a small set of keys carries most mass, which correlates with skew and poor parallel efficiency [38]. If p_k are normalized frequencies, entropy $H = -\sum_k p_k \log p_k$ can be approximated from sketches. Low H suggests that repartitioning by that key may be risky, motivating alternative strategies such as salting, adaptive skew handling, or choosing a different join order that reduces the skewed key's impact.

Communication efficiency is central in distributed SQL. Data movement can dominate runtime, and network is a shared resource. Planning should consider bytes transferred and the number of shuffle boundaries [39]. A useful abstraction is to approximate communication cost by the sum over exchanges of the entropy-coded size of messages. If an intermediate relation has B bytes and compressibility factor $c \in (0, 1]$, then effective transmitted bytes are cB. Compression ratio depends on column encodings and value distributions. Estimating c can be done by sampling or by using statistics such as dictionary cardinality and run-length patterns. From an information-

theoretic viewpoint, the minimum expected bits required to encode values is bounded below by entropy; thus, for a column with entropy H bits per value and n values, transmitted bits are at least nH [40]. While real compressors have overhead, the bound provides intuition for when compression might help. The learned cost model can incorporate empirical compression ratios from telemetry, enabling it to learn that certain intermediate results compress well and therefore may be cheaper to shuffle than their raw size suggests.

Query optimization also interacts with scheduling and admission control. Engines may use fair schedulers, queues, and preemption. Under such policies, the cost surface is non-stationary: the same plan can have different latency depending on queueing delay [41]. A robust optimizer should avoid plans that are extremely resource-hungry even if they are fast in isolation, because they can cause system-wide contention. This is captured by including resource objectives and constraints. One can model the cluster as having budgets for CPU and network per time window and use Lagrangian multipliers to penalize plans that exceed predicted budgets. If $r(\pi)$ is predicted resource usage vector and b is budget, the constrained optimization $\min J_{\text{lat}}(\pi)$ subject to $r(\pi) \le b$ yields KKT-like conditions. In an online setting, the multipliers can be updated via dual ascent:

$$\lambda_{t+1} = \left[\lambda_t + \eta(r(\pi_t) - b)\right]_+,$$

where $[\cdot]_+$ projects to nonnegative values [42]. This resembles congestion control: when resource usage exceeds budget, λ increases and the planner becomes more conservative. Such updates can be implemented without changing the core engine, by modifying the planner's scoring function.

Plan enumeration and search must also be engineered. Classical dynamic programming for join ordering has complexity $O(k^2 2^k)$ for k relations, which is feasible up to moderate k but expensive beyond. In distributed SQL, additional physical properties increase state space [43]. A learned-guided search can reduce enumeration by pruning. However, pruning must be safe. A useful compromise is to run dynamic programming for small k and switch to beam search or iterative deepening for larger k, using learned scores as heuristics. The learned heuristic should be monotone or at least consistent enough to avoid pathological search. While strict A* admissibility is difficult because costs are learned, one can incorporate lower bounds from analytical models to preserve pruning correctness: score a state by $g(s) + h(s)$, where g is known partial cost and h is a learned or analytical estimate of remaining cost, but clamp h below by a conservative analytical bound to reduce the chance of underestimation [44]. This hybrid approach leverages both learned flexibility and analytical safety.

Distributed engines often support adaptive query execution, where plans can change at runtime based on observed cardinalities. This interacts with learning. An end-to-end learned optimizer can be designed to output an adaptive policy rather than a fixed plan, specifying contingencies such as switching from broadcast to shuffle if an intermediate exceeds a threshold. Such a policy can be represented as a decision tree embedded in the plan, or as annotations that enable runtime operators to choose among alternatives [45]. Learning such contingencies resembles learning robust policies under partial observability. The advantage is improved robustness to estimation error and workload shifts. The cost is increased complexity and potential overhead in maintaining alternative paths. A conservative design limits adaptivity to a small number of high-impact switches, particularly around exchanges and join algorithms.

Another systems concern is determinism and debugging [46]. Learned models can introduce non-determinism if they use stochastic sampling during inference. For production, inference should be deterministic given fixed inputs, so sampling can be disabled and argmax used. For exploration, stochasticity can be enabled under controlled conditions. Logging is essential: the optimizer should record features, embeddings, predicted costs, uncertainty, chosen actions, and alternative candidates. This enables offline analysis and replay [47]. To manage storage, logs can be compressed and sampled, and sensitive data such as literals can be anonymized or bucketed.

Finally, deploying learned optimization requires compatibility with query rewriting, security policies, and governance. Some rewrites are mandated, such as enforcing row-level security pred-

icates. These constraints can be integrated into the logical plan before learning. The learned optimizer then operates on an already constrained plan [48]. This avoids unsafe behavior where the model might inadvertently remove required filters. Similarly, resource governance policies can be encoded as constraints. The earlier Lagrangian approach provides a mechanism to enforce such constraints while still allowing learning to optimize within the feasible region.

6 Evaluation Methodology and Reproducibility

Evaluating a learned optimizer under schema and workload shifts requires protocols that separate interpolation from extrapolation. A core principle is that test sets should include queries and schemas that differ structurally from training [49]. For schema shifts, one can evaluate on databases with different numbers of tables, different relationship graphs, and different attribute distributions. For workload shifts, one can evaluate on query templates not seen during training, altered predicate distributions, and different join graph motifs. Because engine behavior depends on hardware and configuration, evaluation should report cluster details, including number of nodes, CPU model, memory, storage type, network bandwidth, and scheduler settings. It should also control for concurrency by running experiments under defined load levels. Results should include not only average latency but also tail metrics, such as 95% and 99% quantiles, because robustness is primarily about avoiding severe regressions [50]. Reporting resource usage, such as network bytes and CPU-seconds, helps interpret whether improvements are achieved by shifting cost to shared resources.

A practical evaluation can compare multiple planners: a baseline rule-based optimizer, a baseline augmented with learned cardinality or cost models, and the end-to-end learned optimizer with robustness mechanisms. Comparisons should include ablations that remove uncertainty penalties, shift detection, online adaptation, and sketch-aware bounds, to isolate their contributions. Because model training can be sensitive to random seeds and data splits, reproducibility requires reporting seeds, using multiple runs, and providing confidence intervals. In addition, training data generation should be described precisely: whether traces are collected from production-like workloads, from synthetic generators, or from enumerated plan runs [51]. If enumerating plans to obtain labels, the exploration policy must be described, since it affects which plans are observed and can bias learning.

Instrumentation is necessary to measure realized costs. Distributed engines provide stage-level metrics: input/output rows, bytes, time, spill events, and shuffle statistics. These metrics can be used to train operator-level cost models and to diagnose failures. For example, if a plan underperforms due to skew, logs should show partition size distributions and straggler tasks [52]. A robust optimizer should reduce the frequency of such pathologies under shift. Evaluation should include stress tests where skew is injected by modifying key distributions, and where histogram staleness is simulated by withholding statistics updates. Such tests approximate production conditions where distributions drift.

When approximate sketches are used, evaluation must measure their overhead and effect on plan quality. Sketch maintenance consumes CPU and memory and may require scanning or sampling [53]. However, sketches can also be updated incrementally. Experiments should report the sketch parameters, such as width and depth for count-min sketches or register count for distinct-count sketches, and measure relative error in estimates. The planner's robustness mechanism that uses confidence intervals should be evaluated by measuring how often it prevents harmful threshold crossings, such as avoiding broadcast when the true size exceeds the broadcast limit. Because sketches trade memory for accuracy, one can evaluate sensitivity to sketch size. A neutral interpretation considers both the runtime benefit from reduced communication and the risk that approximation error leads to worse plans [54].

The evaluation of robustness should explicitly quantify shift. A convenient approach is to define a shift score based on embedding distances or feature divergences and then stratify results by shift score. For example, compute the Mahalanobis distance in embedding space and group queries into bins of low, medium, and high shift. Report performance within each bin. If the learned optimizer degrades gracefully as shift increases, this suggests robustness [55]. If it performs

well only in low-shift bins and collapses in high-shift bins, then robustness mechanisms may be insufficient. Another useful analysis is calibration: compare predicted uncertainty to observed absolute error of cost predictions. Calibration curves can be computed by binning predictions by σ and measuring mean squared error per bin. Better-calibrated uncertainty supports safer decision-making.

Search efficiency is another metric [56]. Learned-guided search should reduce the number of explored states or plans relative to baselines while maintaining or improving plan quality. Report planning time and peak memory. Because planning time can vary with query size, report scaling as a function of number of relations and predicates. A Big-O discussion is useful but insufficient; empirical scaling curves matter. If the learned model uses graph neural networks, report inference time as a function of graph size, and whether caching is used [57]. When caching is used, report hit rates under different workloads, because workload shifts may reduce caching effectiveness.

A careful evaluation also considers failure handling. When the optimizer falls back to baseline due to high uncertainty or detected shift, report how often this occurs and what performance results. Frequent fallback may indicate that the learned component is overly conservative or insufficiently trained for the target domain. Rare fallback with occasional severe regressions may indicate insufficient safety [58]. The trade-off can be tuned via the uncertainty penalty κ and shift thresholds. Reporting these knobs and their effects supports reproducibility and practical deployment decisions.

Reproducibility requires artifacts and deterministic pipelines. Model code should fix random seeds, log hyperparameters, and version dependencies. Data pipelines should record schema snapshots, statistics versions, and cluster configuration [59]. Because distributed execution is noisy, experiments should be repeated and run-order randomized to reduce bias from transient cluster states. When possible, isolated clusters should be used. If isolation is impossible, telemetry should be recorded and used to filter out runs with anomalous interference. For model evaluation, it can be useful to replay query execution in a simulator that approximates engine behavior, enabling large-scale comparisons. However, simulators can diverge from reality, so evaluation should report both simulated and measured results and analyze discrepancies [60]. The cost model can be trained on simulated data and fine-tuned on real traces; evaluation should separate these phases to clarify how much relies on simulation fidelity.

Finally, evaluation should include qualitative analysis of plan changes under shift. For example, when a schema changes by adding an index or changing partitioning, a robust optimizer should adjust exchange decisions accordingly. When workload shifts to more selective predicates, the optimizer may choose different join orders. These behaviors can be examined by comparing plan graphs and operator-level costs [61]. Such analysis helps verify that improvements are not artifacts of measurement noise and can reveal systematic failure modes, such as consistently underestimating costs of wide shuffles or overusing broadcasts under uncertain cardinalities.

7 Conclusion

End-to-end learned query optimization for distributed SQL must reconcile two competing forces: the desire to exploit data-driven models that capture complex cost interactions, and the need for robustness under schema and workload shifts that can invalidate learned correlations. This paper presented a framework that treats planning as structured prediction over physical plans, uses schema- and query-graph embeddings to support variable schemas, and aligns learning objectives with realized execution costs while incorporating constraints on resources and risk. The approach combines learned scoring with search and classical feasibility checks, enabling practical integration into existing optimizers without relinquishing correctness constraints. Differentiable relaxations of discrete decisions and hybrid training strategies connect imitation learning, bandit feedback, and cost-model learning, improving sample efficiency relative to purely reinforcement-based methods in database settings [62].

Robustness was addressed through multiple mechanisms that are complementary rather than ex-

clusive. Schema-invariant representations reduce reliance on table identities and support generalization across evolving schemas. Uncertainty-aware cost prediction and conservative decision rules reduce overconfident threshold crossings, particularly around distribution strategies where errors are costly. Shift detection in embedding space enables adaptive conservatism and controlled fallback. Distributionally robust training and group-aware objectives emphasize tail performance and hard query motifs [63]. Online adaptation with trust regions supports incremental learning under persistent workload changes while mitigating catastrophic forgetting. The integration of approximate sketches and confidence-aware planning ties communication efficiency to bounded estimation error, acknowledging that distributed optimization is inseparable from the cost of maintaining accurate statistics.

Systems integration considerations shaped the design: inference and planning overhead must be controlled via caching, bounded neighborhoods, and hybrid search strategies; distributed execution effects such as skew, critical paths, and contention require cost models that go beyond additive operator sums; and governance and determinism require stable APIs, logging, and safe exploration policies. Evaluation practices that explicitly test across schema and workload shifts, report tail metrics, and document reproducibility are necessary to assess whether a learned optimizer provides predictable behavior rather than benchmark-specific gains. Overall, end-to-end learning can be made compatible with robust distributed SQL optimization when it is embedded within a constraint-aware, uncertainty-aware, and systems-informed architecture. The practical utility depends on careful engineering and evaluation under realistic shift scenarios, and on maintaining safe fallbacks and guardrails that limit the impact of inevitable modeling errors in non-stationary environments [64].

References

[1] T. Heins, R. Glebke, M. Stoffers, S. Gurumurthy, J. Heesemann, M. Josevski, A. Monti, and K. Wehrle, "Delay-aware model predictive control for fast frequency control," in *2023 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–7, IEEE, 10 2023.

[2] *IM - Tofino + P4: A Strong Compound for AQM on High-Speed Networks?*, 5 2021.

[3] A. Heß, F. J. Hauck, and E. Meißner, "Consensus-agnostic state-machine replication," in *Proceedings of the 25th International Middleware Conference*, pp. 341–353, ACM, 12 2024.

[4] W. B. Daszczuk, *Fairness in Distributed Systems Verification*, pp. 139–159. Germany: Springer International Publishing, 3 2019.

[5] R. Malik, S. Kim, X. Jin, C. Ramachandran, J. Han, I. Gupta, and K. Nahrstedt, "Mlr-index: An index structure for fast and scalable similarity search in high dimensions," in *International Conference on Scientific and Statistical Database Management*, pp. 167–184, Springer, 2009.

[6] Y. Tyryshkina, "Understanding join strategies in distributed systems," in *2021 International Seminar on Electron Devices Design and Production (SED)*, pp. 1–4, IEEE, 4 2021.

[7] R. Lichtenthäler and G. Wirtz, "An experimental validation of architectural measures for cloud-native quality evaluations," in *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, pp. 374–384, IEEE, 7 2025.

[8] I. Zhuzhgina and A. Lazarev, "An intelligently distributed system for controlling information flows," *E3S Web of Conferences*, vol. 431, pp. 5017–05017, 10 2023.

[9] N. S. Chatharajupalli, R. Rotta, R. Karnapke, and J. Nolte, "Vibromote: Wi-fi-based mesh communication for railway bridge inspection and monitoring," in *2025 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, IEEE, 10 2025.

[10] R. Kemp, *Devising – Embodied Creativity in Distributed Systems*, pp. 48–57. Routledge, 9 2018.

[11] F. Ansari and A. Afzal, *A Grid-Connected Distributed System for PV System*, pp. 919–924. Springer Singapore, 1 2021.

[12] A. Bolfing, *Distributed Systems*, pp. 143–198. Oxford University PressOxford, 9 2020.

[13] A. Ba, F. O'Donncha, J. Ploennigs, and M. Azmat, "Efficient extraction of insights at the edges of distributed systems," in *2023 IEEE International Conference on Big Data (BigData)*, pp. 1610–1619, IEEE, 12 2023.

[14] R. Müller, S. Langer, F. Ritz, C. Roch, S. Illium, and C. Linnhoff-Popien, *PKDD/ECML Workshops (2) - Soccer Team Vectors.*, pp. 247–257. Germany: Springer International Publishing, 3 2020.

[15] T. He and R. Buyya, "A taxonomy of live migration management in cloud computing," *ACM Computing Surveys*, vol. 56, pp. 1–33, 10 2023.

[16] E. Becks, P. Zdankin, V. Matkovic, and T. Weis, "Complexity of smart home setups: A qualitative user study on smart home assistance and implications on technical requirements," *Technologies*, vol. 11, pp. 9–9, 1 2023.

[17] Y. Braidiz, D. Efimov, A. Polyakov, and W. Perruquetti, "On robustness of finite-time stability of homogeneous affine nonlinear systems and cascade interconnections," *International Journal of Control*, pp. 1–11, 9 2020.

[18] E. E. Mohammed, R. Y. S. Naji, A. A. Hussein, M. A. Saeed, and R. A. M. A. Selwi, "Anomaly detection system for secure cloud computing environment using machine learning," in *2025 5th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, pp. 1–9, IEEE, 8 2025.

[19] R. Chandrasekar, R. Suresh, and S. Ponnambalam, "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 International Conference on Advanced Computing and Communications*, pp. 628–629, IEEE, 2006.

[20] R. Kuznets, "Communication modalities," 5 2024.

[21] J. D. Herath, P. Yang, and G. Yan, "Codaspy - real-time evasion attacks against deep learning-based anomaly detection from distributed system logs," in *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, pp. 29–40, ACM, 4 2021.

[22] J. Flak, T. Skowron, R. Cupek, M. Fojcik, D. Caban, and A. Domański, "Zigbee network for agv communication in industrial environments," in *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–9, IEEE, 10 2023.

[23] M. Breyer, A. V. Craen, and D. Pflüger, "A comparison of sycl, opencl, cuda, and openmp for massively parallel support vector machine classification on multi-vendor hardware," in *International Workshop on OpenCL*, pp. 1–12, ACM, 5 2022.

[24] T. Pusztai, C. Marcelino, and S. Nastic, "Hyperdrive: Scheduling serverless functions in the edge-cloud-space 3d continuum," in *2024 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 265–278, IEEE, 12 2024.

[25] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and A. Manikandan, "Localized tree change multicast protocol for mobile ad hoc networks," in *2006 International Conference on Wireless and Mobile Communications (ICWMC'06)*, pp. 44–44, IEEE, 2006.

[26] F. Strnisa, M. Jancic, and G. Kosec, "A meshless solution of a small-strain plasticity problem," in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 257–262, IEEE, 5 2022.

[27] G. Farina, "Tractable reliable communication in compromised networks," 12 2020.

[28] Z. J. Hamad and S. R. M. Zeebaree, "Recourses utilization in a distributed system: A review," 10 2021.

[29] D. T. Dang and D. Hwang, "Consensus-based methods for distributed systems, blockchain, and voting: a survey," *Journal of Information and Telecommunication*, pp. 1–24, 10 2024.

[30] R. Rotta, J. Schulz, B. Naumann, N. S. Chatharajupalli, J. Nolte, and M. Werner, "B.a.t.m.a.n. mesh networking on esp32's 802.11," in *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, vol. 3, pp. 1–7, IEEE, 10 2024.

[31] Y. Tytarchuk, S. Pakhomov, D. Beirak, V. Sydorchuk, and S. V. Zaitseva, "The impact of distributed systems on the architecture and design of computer systems: advantages and challenges," *Data and Metadata*, vol. 3, 12 2024.

[32] F. Neves, R. Vilaca, and J. Pereira, "Detailed black-box monitoring of distributed systems," *ACM SIGAPP Applied Computing Review*, vol. 21, pp. 24–36, 7 2021.

[33] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, pp. 2701–2706, 2007.

[34] S. Hussain, A. Sajjad, and Z. Javed, "Deadlock detection in distributed system," 1 2020.

[35] E. Becks, M. Josten, V. Matkovic, and T. Weis, "Revising poor man's eye tracker for crowdsourced studies," in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 328–330, IEEE, 3 2023.

[36] A. Furutanpey, P. A. Frangoudis, P. Szabo, and S. Dustdar, "Adversarial robustness of bottleneck injected deep neural networks for task-oriented communication," in *2025 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, pp. 1–6, IEEE, 5 2025.

[37] A. Fentis, C. Lytridis, V. G. Kaburlasos, E. Vrochidou, T. Pachidis, E. Bahatti, and M. Mestari, "A machine learning based approach for next-day photovoltaic power forecasting," in *2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS)*, pp. 1–8, IEEE, 10 2020.

[38] J. Yuan, A. Le-Tuan, M. Hauswirth, and D. Le-Phuoc, "Cooperative students: Navigating unsupervised domain adaptation in nighttime object detection," in *2024 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, IEEE, 7 2024.

[39] *A Generic Review of Messenger Application: WeChat and WhatsApp*, vol. 1, 12 2020.

[40] G. Neumann, P. Grace, D. Burns, and M. Surridge, "Pseudonymization risk analysis in distributed systems," *Journal of Internet Services and Applications*, vol. 10, pp. 1–16, 1 2019.

[41] M. Witter and A. R. D. Vit, "Blockchain e sistemas distribuídos: conceitos básicos e implicações," 3 2024.

[42] *AAMAS - Achieving Sybil-Proofness in DistributedWork Systems*, 9 2021.

[43] E. M. D. Souza, N. Suzin, C. A. Zeferino, and D. R. Melo, "Interactive simulator for risc-v assembly programming," in *2025 17th Seminar on Power Electronics and Control (SEPOC)*, pp. 1–8, IEEE, 11 2025.

[44] C. Chenavier and M. Lucas, "The diamond lemma for non-terminating rewriting systems using deterministic reduction strategies," 6 2019.

[45] M. Sun, Y. Teng, F. Zhao, J. Qi, D. Jiang, and C. Fan, *Spatio-Textual Group Skyline Query*, pp. 34–50. Germany: Springer Nature Switzerland, 9 2023.

[46] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–6, IEEE, 2006.

[47] "Three-layer distributed system based on bayesian classifier," *Distributed Processing System*, vol. 3, 10 2022.

[48] C. Hanane, A. Battou, and O. Baz, "Performance security in distributed system: Comparative study," *International Journal of Computer Applications*, vol. 179, pp. 29–33, 1 2018.

[49] I.-A. Secara, *Challenges and Considerations in Developing and Architecting Large-scale Distributed Systems*, pp. 1–15. B P International (a part of SCIENCEDOMAIN International), 4 2023.

[50] Z. Dong, C. Tang, J. Wang, Z. Wang, H. Chen, and B. Zang, "Optimistic transaction processing in deterministic database," *Journal of Computer Science and Technology*, vol. 35, pp. 382–394, 3 2020.

[51] M. Jančič, M. Rot, and G. Kosec, *Spatially-Varying Meshless Approximation Method for Enhanced Computational Efficiency*, pp. 500–514. Germany: Springer Nature Switzerland, 6 2023.

[52] R. Laidig, F. Shibli, B. Tufekci, F. Dürr, and C. Tunc, "Improving drone communication qos through adaptive redundancy," in *2025 34th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–9, IEEE, 8 2025.

[53] *Scalable Distributed Systems*, pp. 2290–2290. Springer New York, 6 2018.

[54] *On finite-time stability analysis of homogeneous Persidskii systems using LMIs*, 11 2021.

[55] C. Deng, Z. Shen, D. Li, Z. Mi, and Y. Xia, "The design and optimization of memory ballooning in sev confidential virtual machines," in *2024 IEEE International Conference on Joint Cloud Computing (JCC)*, pp. 9–16, IEEE, 7 2024.

[56] M. Zakarya, L. Gillam, K. Salah, O. F. Rana, S. Tirunagari, and R. BUYYA, "Colocateme: Aggregation-based, energy, performance and cost aware vm placement and consolidation in heterogeneous iaas clouds," 6 2021.

[57] A. Arman, P. Bellini, D. Bologna, P. Nesi, G. Pantaleo, and M. Paolucci, "Automating iot data ingestion enabling visual representation.," *Sensors (Basel, Switzerland)*, vol. 21, pp. 8429–8429, 12 2021.

[58] R. Ângelo Santos Filipe, "Client-side monitoring of distributed systems," 2 2020.

[59] L. Guegan, B. L. Amersho, A.-C. Orgerie, and M. Quinson, *AINA - A Large-Scale Wired Network Energy Model for Flow-Level Simulations*, vol. 926, pp. 1047–1058. Springer International Publishing, 3 2019.

[60] L. Su, X. Wang, and L. Wang, "A resilience analysis method for distributed system based on complex network," in *2021 IEEE International Conference on Unmanned Systems (ICUS)*, pp. 238–243, IEEE, 10 2021.

[61] R. Malik, C. Ramachandran, I. Gupta, and K. Nahrstedt, "Samera: a scalable and memory-efficient feature extraction algorithm for short 3d video segments.," in *IMMERSCOM*, p. 18, 2009.

[62] B. K. Ozkan, R. Majumdar, F. Niksic, M. T. Befrouei, and G. Weissenbacher, "Randomized testing of distributed systems with probabilistic guarantees," *Proceedings of the ACM on Programming Languages*, vol. 2, pp. 160–28, 10 2018.

[63] X. Song, R. Chen, H. Song, Y. Zhang, and H. Chen, "Unified and near-optimal multi-gpu cache for embedding-based deep learning," *ACM Transactions on Computer Systems*, vol. 44, pp. 1–32, 11 2025.

[64] K. M. Goeschka, R. P. S. de Oliveira, P. Pietzuch, and G. Russello, "Session details: Theme: Distributed systems: Dads - dependable, adaptive, and secure distributed systems track," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, ACM, 4 2019.