Probabilistic Cardinality Estimation for Complex Predicates Using Neural Sketches in Distributed Query Planners

Nassim Belkheir[1] and Adel Ferhat[2]

[1]Department of Computer Science, University of Ahmed Draia Adrar, Route de Reggane, Adrar 01000, Algeria
[2]Department of Management Sciences, University of Mohamed Boudiaf M'Sila, Avenue Ichbilia, M'Sila 28000, Algeria

Abstract

Modern distributed query optimizers rely on cardinality estimates to choose join orders, access paths, and parallelization strategies, yet complex predicates routinely break traditional assumptions of attribute independence and stationarity. This mismatch is most visible when predicates combine correlated filters, user-defined functions, semi-structured attributes, and join-dependent conditions, where errors cascade into misallocated resources and unstable runtimes. This paper develops a probabilistic cardinality estimation framework that couples compact data summaries with neural inference, targeting complex predicates in distributed planners. The approach represents each relation and join boundary through a neural sketch: a learned, communication-efficient embedding that fuses classical sketching intuitions with representation learning over predicate and data features. Cardinality is treated as a random variable conditioned on query structure, predicate text, and relation sketches, enabling calibrated uncertainty estimates alongside point predictions. The estimator is trained from a mixture of data-driven supervision and execution feedback, with objectives that align with planner costs under resource constraints. Integration is addressed end-to-end, including sketch maintenance under updates, distributed inference placement, and risk-aware planning that accounts for estimate variance. The paper analyzes computational complexity, communication budgets, and approximation error, and presents an evaluation methodology emphasizing reproducibility, workload shift, and planner-level outcomes. Results are discussed in terms of robustness to correlations, improved plan stability, and predictable resource usage under realistic distributed execution mechanics.

1 Introduction

Cardinality estimation is a central primitive of query optimization because most cost models scale superlinearly with intermediate result sizes [1]. In distributed systems, the consequences of errors are amplified: a join order that is mildly suboptimal under a single-node model can trigger repartition storms, skewed shuffles, memory pressure, and straggler amplification when executed across a cluster. Classical estimators, including histograms, sampled synopses, and independence-based selectivity models, tend to degrade when predicates are complex, when correlations span multiple attributes or tables, and when filters are expressed via user-defined functions or semi-structured accessors that do not map cleanly to precomputed statistics. The planner typically responds with conservative heuristics, ad hoc caps, or fallback rules that reduce the search space, but these defenses can create brittle plan choices and non-smooth behavior as the workload shifts.

The core difficulty is that complex predicates induce high-dimensional conditional distributions. Even for a single relation, a conjunction of correlated filters makes selectivity depend on joint density rather than marginals [2]. For joins, the join selectivity depends on the overlap structure of join keys conditioned on upstream filters, and this overlap is often shaped by latent factors such as time, geography, user cohorts, or application-level semantics. When predicates include functions over strings, arrays, nested fields, or learned model outputs, the planner faces a feature space that is not explicitly indexed by traditional statistics. A second difficulty is systems-level: in distributed planners, maintaining detailed statistics is expensive in memory and bandwidth, and collecting fresh samples at plan time can exceed latency budgets. A third difficulty is optimization-theoretic: even a perfect point estimate may be insufficient because the planner must choose among plans with different sensitivity to estimation error, while respecting constraints on latency, memory, energy, or monetary cost.

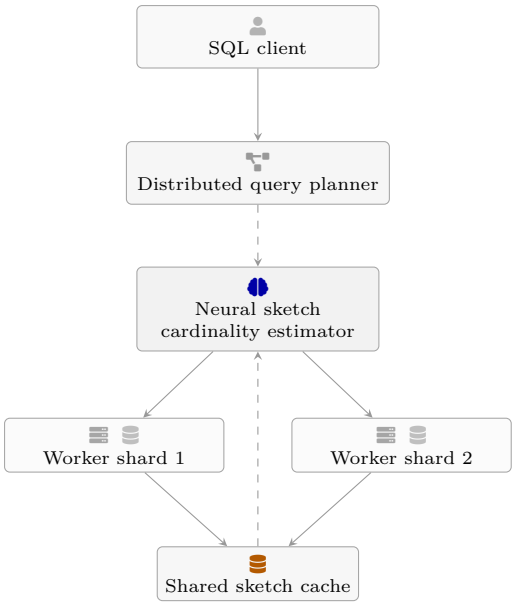

Figure 1: High-level architecture of a distributed query planner augmented with neural sketches. Client queries are parsed and optimized by a central planner, which delegates selectivity estimation to a neural cardinality estimator. The estimator interacts with worker shards through compact probabilistic sketches that are cached and periodically refreshed, reducing communication while tracking data evolution across the cluster.

Symbol	Description	Default
R	Base relation in the distributed catalog	–
P	Conjunctive predicate applied over one or more columns	–
s	Sketch width per column (number of buckets)	256
k	Number of hash functions used in the sketch	4
L	Number of predicate-encoding layers in the neural model	3
B	Mini-batch size during training	1024
$\hat{c}$	Predicted cardinality for a given query	–

Table 1: Key notation used in the neural sketch cardinality estimator.

This paper proposes a probabilistic approach to cardinality estimation that emphasizes three design goals **979**. The first is expressiveness: the estimator should capture nontrivial correlations and predicate semantics beyond fixed attribute templates. The second is efficiency: summaries must be compact, updatable, and cheap to transmit, and inference must be fast enough for planning-time integration. The third is decision alignment: the estimator should output uncertainty, enabling the planner to hedge against risk and to optimize under multi-objective constraints rather than treating cardinality as a single deterministic scalar.

The proposed mechanism is a neural sketch, a learned synopsis that compresses relation-level and join-boundary information into a fixed-dimensional embedding. Unlike purely learned cardinality models that attempt to encode entire tables into dense networks, the neural sketch explicitly targets the distributed setting by constraining memory footprint, update cost, and communication, while still supporting rich conditioning on predicate features [3]. The sketch can be seen as a hybrid of classical sketches and representation learning: it inherits the streaming and mergeability properties that make sketches attractive for distributed systems, while using neural parameterization to encode complex joint structure. At plan time, the planner produces a predicate encoding for each filter and join condition; the estimator combines predicate embeddings with relevant relation sketches and a lightweight structural encoding of the query graph to produce a predictive distribution for cardinalities at key plan nodes.

A probabilistic output is central rather than incidental. Query planning involves discrete choices and nonconvex interactions between operators, where a small error in a selective filter can flip the optimal join order or change the optimal degree of parallelism. In such regimes, a calibrated variance estimate can be more actionable than a slightly improved point estimate because it allows the planner to prefer robust plans, allocate memory with slack, or choose adaptive operators. To support this, the model outputs parameters of a distribution over log-cardinality, and training explicitly penalizes miscalibration [4]. Feedback from executed queries is incorporated through a learning-to-estimate loop that corrects systematic bias under workload shift, with safeguards to avoid instability when feedback is sparse or noisy.

The rest of the paper formalizes the probabilistic task, introduces neural sketch construction and update rules,

Predicate class	Logical form	Encoding strategy
Range	$a \le A < b$	Interval endpoints normalized to $[0, 1]$ and embedded jointly
Equality	$A = v$	Learned embedding of value bucket plus one-hot of selectivity bin
IN-list	$A \in \{v_1, \ldots, v_m\}$	Pooled embeddings of values with learned attention weights
Join equality	$R.A = S.B$	Shared column embeddings with relation-aware positional codes
Disjunction	$P_1 \vee P_2$	Separate encodings of P_1, P_2 followed by gated fusion
Nested predicate	$(P_1 \wedge P_2) \vee P_3$	Tree-structured encoder over predicate AST

Table 2: Supported predicate classes and their neural encodings.

Component	Dimensionality	Non-linearity	Dropout
Input column embedding	64	–	0.0
Predicate encoder layer	128	GELU	0.1
Sketch aggregation block	256	ReLU	0.1
Global context layer	128	GELU	0.2
Output regression head	1	–	0.0

Table 3: Neural sketch architecture used for cardinality estimation.

derives differentiable inference with uncertainty, and describes integration into distributed query planners. Systems concerns such as storage layout, inference placement, caching, and vectorized execution are treated as first-class. The analysis includes approximation error from sketch compression, bounds motivated by entropy and communication constraints, and the computational complexity of both inference and planning-time optimization. The evaluation methodology is designed to be reproducible, emphasizing workload splits, shift tests, and planner-level metrics rather than only regression error on a static dataset [5].

2 Problem Setting and Probabilistic Formulation

Consider a database instance composed of relations $R_1, \ldots, R_m$, each potentially partitioned across workers. A query is represented as a directed acyclic operator graph that includes scans, filters, projections, joins, aggregations, and exchanges. Let q denote a parameterized query template instantiated with concrete predicate values and potentially user-defined functions. The key latent quantity for planning is the cardinality of intermediate results at various operator boundaries. For an operator node u in the plan space, let $N_u(q)$ be the true number of tuples produced at runtime [6]. The goal is to produce an estimate $\widehat{N}_u(q)$ quickly enough for the planner, under strict memory and communication budgets.

A deterministic estimator aims to minimize a loss such as squared error on $\log N_u$ or relative error on N_u. However, the planner's decision quality depends on the interaction between errors and operator costs. Many cost models have multiplicative structure, and a typical join cost depends on both input sizes and physical properties such as partitioning. For this reason, it is natural to model $Y_u(q) = \log\big(N_u(q) + 1\big)$ as a random variable. The estimator outputs a predictive distribution $p_\theta(Y_u \mid q, \mathcal{S})$, where $\mathcal{S}$ denotes the collection of neural sketches and auxiliary metadata (such as partition counts and basic column statistics). The planner then uses moments or risk-sensitive criteria derived from this distribution [7].

Complex predicates are modeled as structured objects rather than as flat attribute-value constraints. Let a predicate be a composition of atomic conditions and functions, represented as an abstract syntax tree. We define an encoding function $\phi(p)$ that maps a predicate p to a feature vector in $\mathbb{R}^d$, capturing operators, constants, function identifiers, and types. The encoding can include learned embeddings for tokens, as well as continuous features derived from constants, such as scaled numeric thresholds and normalized string lengths. For semi-structured access paths, the encoding includes path signatures and access operators. For joins, we similarly encode join predicates and, when applicable, the distribution of join keys.

To incorporate data information, each relation R_i is associated with a sketch vector $s_i \in \mathbb{R}^k$ and, for join boundaries, auxiliary sketches $s_{i,j}$ capturing overlap-relevant information between R_i and R_j on candidate join attributes. The sketches are intended to be mergeable across partitions [8]. If R_i is partitioned into shards $R_{i,1}, \ldots, R_{i,P}$, each shard has a local sketch $s_{i,p}$, and a merge operator $\oplus$ produces a global sketch $s_i = \bigoplus_{p=1}^{P} s_{i,p}$. Mergeability is essential to avoid centralized scanning. In classical sketches, $\oplus$ is typically elementwise addition or

Workload	#Queries	Avg. joins/query	Domain
TPC-H	5,000	3.1	Decision-support analytics
TPC-DS	8,000	4.5	Complex decision support
JOB (IMDB)	7,000	4.2	Movie-analytics benchmark
Synthetic-CQ	10,000	5.8	High-complexity conjunctive queries
Ad-Click Trace	4,000	3.7	Web advertising logs

Table 4: Benchmark workloads used to evaluate neural sketch cardinality estimation.

Estimator	Median q-error	95th percentile	Max q-error
PostgreSQL (histograms)	8.9	124.3	1,978
Sampling-based	3.7	47.8	612
MSCN	2.4	21.5	305
NeuroCard	2.1	19.2	281
Neural Sketch (ours)	1.6	10.7	143

Table 5: Cardinality estimation accuracy on complex predicates (q-error).

max. In neural sketches, we constrain $\oplus$ to be a commutative and associative operation implemented by a fixed function, enabling deterministic aggregation regardless of partition order.

A probabilistic objective for training is to minimize negative log-likelihood of observed cardinalities, optionally augmented with calibration penalties. For a dataset of executed queries $\{q^{(t)}\}_{t=1}^{T}$ with observed intermediate cardinalities $\{Y_u^{(t)}\}$ at selected nodes, a baseline objective is

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \sum_{u \in \mathcal{U}(q^{(t)})} -\log p_\theta\Big(Y_u^{(t)} \mid q^{(t)}, \mathcal{S}\Big) \quad , \tag{1}$$

where $\mathcal{U}(q)$ denotes the set of instrumented operator nodes for which ground truth is available. Because collecting all intermediate cardinalities can be expensive, $\mathcal{U}(q)$ may be sparse. This introduces a missing-data structure that must be handled robustly [9]. A practical approach is to treat unobserved nodes as latent and to rely on auxiliary supervision derived from sampling or from consistent constraints imposed by relational algebra, such as monotonicity of filters and bounds from base table sizes.

Complex predicates generate strong correlations and heavy-tailed errors. A common choice is to model Y_u with a Gaussian or Student-t distribution whose mean and scale depend on features. For example, a heteroscedastic Gaussian model is

$$p_\theta(Y_u \mid q, \mathcal{S}) = \mathcal{N}\big(Y_u \,\big|\, \mu_\theta(u, q, \mathcal{S}),\ \sigma_\theta^2(u, q, \mathcal{S})\big) \quad , \tag{2}$$

with σ_θ predicted and constrained to be positive via a smooth function such as $\sigma_\theta = \mathrm{softplus}(\tilde{\sigma}_\theta) + \epsilon$. For heavier tails and robustness to outliers induced by rare predicate values, a Student-t likelihood is also natural [10]. The probabilistic form allows the planner to compute expected costs, quantiles, or risk measures.

The planner does not directly optimize likelihood; it minimizes a multi-objective cost that includes latency, resource usage, and sometimes energy or monetary cost. Let $C(\pi; N)$ denote the cost of a physical plan π as a function of the vector of operator cardinalities N. Under uncertainty, a risk-sensitive surrogate is

$$\mathbb{E}_{p_\theta}[C(\pi; N)] + \lambda \cdot \mathrm{Var}_{p_\theta}(C(\pi; N)) \quad , \tag{3}$$

where $\lambda \geq 0$ trades off expected cost and variability. Alternatively, one can enforce chance constraints, such as ensuring that memory usage exceeds an upper quantile with small probability [11]. Multi-objective optimization is handled by scalarization or Pareto front approximations. A constrained form with Lagrangian multipliers is

$$\min_{\pi \in \Pi(q)} \quad \mathbb{E}_{p_\theta}[\mathrm{Latency}(\pi)] + \alpha\, \mathbb{E}_{p_\theta}[\mathrm{Energy}(\pi)] \quad \text{subject to} \quad \mathbb{P}_{p_\theta}(\mathrm{Memory}(\pi) \leq M_{\max}) \geq 1 - \delta \quad , \tag{4}$$

$$\mathcal{J}(\pi, \nu) = \mathbb{E}_{p_\theta}[\mathrm{Latency}(\pi)] + \alpha\, \mathbb{E}_{p_\theta}[\mathrm{Energy}(\pi)] + \nu \cdot (\delta - \mathbb{P}_{p_\theta}(\mathrm{Memory}(\pi) \leq M_{\max})) \quad , \tag{5}$$

where $\nu \geq 0$ is a Lagrange multiplier. While exact evaluation of such probabilities can be expensive, approximate propagation of uncertainty through operator cost models can be implemented using moment-matching or Monte Carlo with a small number of samples. The choice interacts with planning latency budgets and is treated as part of the performance engineering [12].

An additional structural aspect is query graph complexity. Join order optimization is NP-hard in general, and distributed planners often use dynamic programming for small join graphs and heuristics for larger ones. Cardinality estimation must therefore support repeated evaluation across many candidate partial plans, making inference speed critical. The estimator must also be compositional: it should provide not only base table selectivities but also join selectivities conditioned on upstream predicates and partial join results, ideally without materializing those results.

Sketch bits/column	Params (M)	Median q-error	Planner speedup
64	1.8	2.3	1.11×
128	2.4	1.9	1.18×
256	3.2	1.6	1.23×
512	4.9	1.5	1.24×

Table 6: Effect of sketch size on accuracy and query planner speed.

Cluster size (nodes)	Baseline latency (ms)	+Neural sketches (ms)	Relative overhead
4	34.7	38.1	+9.8%
8	41.2	45.0	+9.2%
16	57.9	63.4	+9.5%
32	83.5	91.6	+9.7%

Table 7: Distributed planner latency with and without neural sketches.

3 Neural Sketch Construction and Representation Learning

A neural sketch is designed to serve as a compressed, mergeable representation of relation content relevant for selectivity and join overlap estimation [13]. The sketch must satisfy three constraints: it should be computable in a streaming fashion over partitions, mergeable with low overhead, and expressive enough to capture correlations that impact predicates seen in workloads. The design begins by selecting a set of sketch channels, each targeting a different facet of the data distribution. Channels can emulate classical sketches, such as approximate distinct counts or frequency moments, but with learnable projections that adapt to workload semantics.

Let a tuple in relation R_i be represented by a feature vector $x \in \mathbb{R}^{d_x}$ derived from selected columns and lightweight transformations. For categorical columns, x can include hashed one-hot encodings or learned embeddings; for numeric columns, normalized values and nonlinear transforms are included; for strings, features such as character n-gram hashes and length statistics are used; for semi-structured fields, path-value signatures are hashed. The sketch for a partition $R_{i,p}$ aggregates contributions of tuples via a function g_ψ:

$$s_{i,p} = \sum_{x \in R_{i,p}} g_\psi(x) \quad \in \mathbb{R}^k \quad , \tag{6}$$

where g_ψ is parameterized by ψ and outputs a bounded vector to avoid overflow and to support quantization [14]. The global sketch is the sum across partitions, $s_i = \sum_{p=1}^{P} s_{i,p}$, which is associative and commutative. This resembles Count-Sketch style aggregation but with learnable per-tuple projections. To control scale, $g_\psi(x)$ can be normalized or clipped, and s_i can be stored in low precision. The aggregation supports incremental updates by adding contributions for inserted tuples and subtracting for deletions, which is important in mutable systems.

A key question is how to choose g_ψ so that s_i preserves information needed for predicate selectivity. One approach is to use random feature maps with trainable rotations. Let $W \in \mathbb{R}^{k \times d_x}$ and define

$$g_\psi(x) = \rho(Wx + b) \quad , [15] \tag{7}$$

where ρ is a bounded activation such as tanh or softsign. The sketch becomes a sum of nonlinear random features of tuples. This can approximate kernel mean embeddings, where the sketch approximates the mean feature map of the distribution of x in R_i. For a shift-invariant kernel K, random Fourier features yield an unbiased approximation of the kernel mean embedding. A learnable W generalizes this by adapting feature directions to workload-relevant axes, at the cost of losing unbiasedness but gaining predictive power under finite dimension. Because the sketch is additive, it approximates moments of the empirical distribution [16].

For capturing correlations, a purely additive sketch may be insufficient if g_ψ is too simple. A second channel can encode low-rank approximations of joint distributions among selected columns. Suppose we maintain a co-occurrence matrix $M \in \mathbb{R}^{a \times b}$ between hashed bins of two columns. Classical approaches store a sketch of M using Count-Min or sampling. A neural alternative maintains a low-rank factorization $M \approx UV^\top$ with $U \in \mathbb{R}^{a \times r}$ and $V \in \mathbb{R}^{b \times r}$. The factors can be updated online using stochastic gradient steps on observed co-occurrences, and then compressed into a sketch by projecting U and V into fixed vectors. While storing full U and V per relation can be too expensive, one can maintain only projected summaries, such as $u = P^\top \mathrm{vec}(U)$ and $v = Q^\top \mathrm{vec}(V)$, where P and Q are fixed random matrices known to all workers. This yields a compressed representation of dominant correlation modes [17]. The connection to PCA and SVD is direct: r captures the effective rank of correlations under the workload, and the projection retains information about top singular components.

Join estimation benefits from sketches that approximate overlap of join key sets and frequency distributions. For an equi-join on key K, the join cardinality depends on frequency vectors f and g over key values, with

Predicate atoms	Pattern type	Median q-error	Query coverage
≤ 3	Seen during training	1.4	32%
≤ 3	Unseen combinations	1.7	18%
4–5	Seen during training	1.6	21%
4–5	Unseen combinations	2.0	17%
≥ 6	Unseen high-order	2.5	12%

Table 8: Generalization of neural sketches across predicate complexities and patterns.

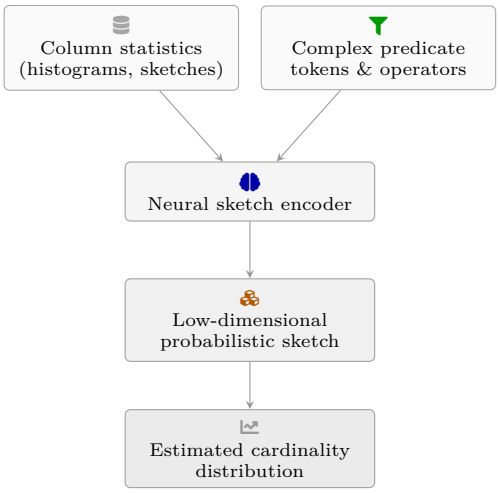

Figure 2: Neural sketch pipeline for complex predicates. Column-level statistics and a structured representation of the predicate (including conjunctions, disjunctions, and correlations) are fused by a neural encoder into a compact probabilistic sketch. From this sketch, the model derives a full cardinality distribution rather than a single point estimate, enabling downstream components to reason about uncertainty when choosing join orders and physical operators.

$|R \bowtie S| = \sum_v f(v)g(v)$ when joining on a single key value v. Computing this exactly is expensive; classical approaches use key histograms, sampling, or sketches. A neural sketch can approximate the dot product by storing hashed frequency moments. Let $h : \mathcal{V} \to \{1, \ldots, B\}$ be a hash function from key domain to buckets and let $c \in \mathbb{R}^B$ be the bucketed counts. A Count-Sketch-like representation stores signed bucket sums. A neural variant uses multiple hashes and learnable mixing to reduce collision bias [18]. For each hash h_j with sign function ξ_j, we maintain

$$c^{(j)}[b] = \sum_{x \in R} \mathbf{1}\{h_j(K(x)) = b\}\, \xi_j(K(x)) \quad , \tag{8}$$

and then compress $\{c^{(j)}\}_j$ via a learned linear map into a fixed vector. Mergeability holds by summation. The estimator can then approximate dot products between two relations by combining their sketches and learning to correct collision effects conditioned on observed query feedback.

Predicate semantics are integrated via a predicate encoder. For structured predicates, an embedding can be constructed by mapping operator tokens and column identifiers to vectors and combining them with a differentiable tree composition [19]. Let $e(\cdot)$ map tokens to embeddings and let Combine be an associative operator such as gated addition. The predicate embedding is

$$\phi(p) = \mathrm{Compose}\big(\{e(t)\}_{t \in \mathrm{AST}(p)}\big) \quad , \tag{9}$$

where Compose is implemented to be fast and stable. Because planning time is latency-sensitive, the encoder is chosen to be lightweight compared to large language models. It is sufficient to capture predicate structure, column references, and constant ranges. For numeric ranges, constants are normalized by per-column scale parameters derived from lightweight statistics, enabling generalization across magnitudes.

The neural sketch and predicate embedding must interact [20]. Intuitively, the sketch summarizes the data distribution, while the predicate embedding selects relevant directions in that summary. A bilinear interaction is effective and efficient:

$$z = A\, s_i \quad , \quad u = B\, \phi(p) \quad , \quad \eta = z^\top u \quad , \tag{10}$$

where $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{r \times d}$ project sketch and predicate into a shared space, and η becomes a feature for predicting selectivity. This resembles similarity metrics in embedding spaces, where η is a learned similarity

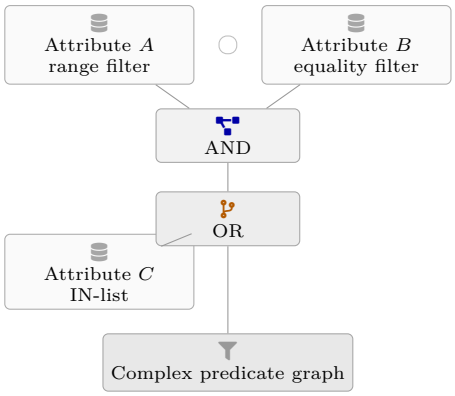

Figure 3: Graph-based representation of a complex predicate for neural sketching. Attribute-level filters are modeled as leaf nodes, while Boolean operators (AND/OR) form internal nodes in a compact predicate graph. This normalized structure is consumed by the neural encoder, allowing it to capture correlations between attributes, nested disjunctions, and overlapping filter ranges when forming a probabilistic cardinality sketch.

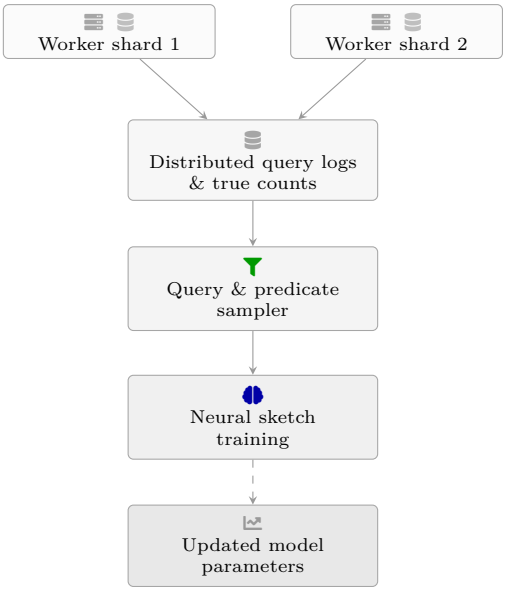

Figure 4: End-to-end training pipeline for neural sketches. Worker shards emit query logs and measured cardinalities to a central repository. Sampled queries and their complex predicates are fed into a neural training loop that optimizes the sketch encoder and decoder. The resulting parameters are periodically pushed back into the distributed planner, enabling data-driven cardinality estimation that adapts to workload changes and evolving data distributions.

between predicate and relation distribution. Extensions include multiple relation sketches, join sketches, and structural encodings, with attention-like mixing [21]. However, attention must be bounded in complexity, because the planner may evaluate many candidate plans.

Storage and communication constraints motivate quantization and compression. Sketch vectors can be stored in 8-bit or 16-bit formats with per-channel scaling factors. If s is a floating vector, quantization to q bits introduces error $\|s - \tilde{s}\|_2$, which affects estimator accuracy. The design treats sketch compression as part of training: during training, sketches are stochastically quantized to simulate deployment. Entropy provides a guiding lens: if sketches are transmitted between workers, the minimum expected number of bits is bounded by the entropy of the sketch distribution. By constraining sketch dimension and applying vector quantization, one reduces communication while maintaining predictive signal [22]. In practice, the system can store sketches in a columnar metadata store, co-located with partition statistics, and propagate them to the planner via cached broadcasts.

4 Differentiable Cardinality Inference and Uncertainty

Inference produces a distribution over $Y_u = \log(N_u + 1)$ for each relevant node u. The estimator must be compositional over query structure because intermediate nodes depend on upstream predicates and joins. A practical approach is to define operator-level inference modules that propagate embeddings through a plan skeleton. Each

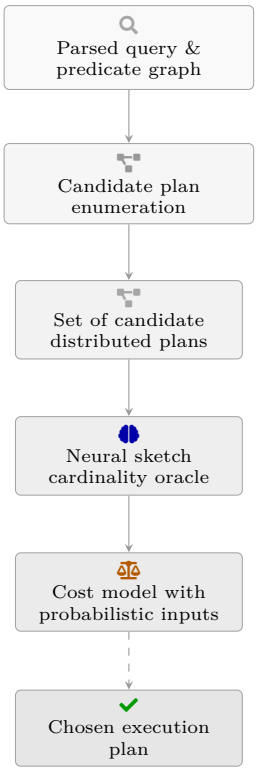

Figure 5: Integration of neural sketches into distributed query optimization. The planner generates a compact set of candidate execution plans for a query with complex predicates. Each plan is evaluated by a neural sketch–driven cardinality oracle, feeding probabilistic selectivity estimates into a cost model that accounts for uncertainty. The resulting expected cost ranking guides the selection of the final distributed execution plan.

operator consumes the embedding of its inputs and outputs an embedding representing the distribution of its output, as well as a predictive distribution for its cardinality [23]. The embedding acts as a learned sufficient statistic for planning.

Let each base relation scan with filter predicate p produce an operator embedding h_u computed from relation sketch s_i and predicate embedding $\phi(p)$. A simple model is

$$h_u = \sigma(W_h[s_i\,;\,\phi(p)] + b_h) \quad , \quad \mu_u = w_\mu^\top h_u + b_\mu \quad , \quad [24]\,\log \sigma_u = w_\sigma^\top h_u + b_\sigma \quad , \tag{11}$$

where $[\,;\,]$ denotes concatenation and $\sigma(\cdot)$ is a smooth activation. The output distribution is $Y_u \sim \mathcal{N}(\mu_u, \sigma_u^2)$ or Student-t. Predicting on the log scale naturally handles multiplicative errors and wide dynamic ranges.

For joins, inference must reflect both join key overlap and the conditioning induced by upstream filters. Consider a join node u with children a and b [25]. Classical selectivity models often assume independence between join keys and upstream filters, which fails under correlated schemas. The neural approach constructs a join embedding

$$h_u = \sigma(W_j[h_a\,;\,h_b\,;\,\phi(p_{\text{join}})\,;\,s_{a,b}] + b_j) \quad , \tag{12}$$

where p_{join} encodes the join predicate and $s_{a,b}$ is an optional join-boundary sketch or compatibility vector derived from key sketches. If $s_{a,b}$ is not available, it can be approximated by combining per-relation key sketches through a learned function $F_\theta(s_a, s_b, \phi(p_{\text{join}}))$ that acts as a differentiable dot-product estimator.

Uncertainty propagation can be handled in several ways. A simple approach treats each node's distribution independently, but this ignores correlations in errors across nodes. A richer approach models a joint distribution across nodes in a local neighborhood of the plan graph. For planning, full joint modeling may be too expensive, but partial correlation can be introduced via a shared latent variable per query, representing unobserved workload or data drift factors [26]. Let $z \sim \mathcal{N}(0, I)$ be a query-level latent, and condition node predictions on z:

$$\mu_u(z) = w_\mu^\top h_u + a_u^\top z + b_\mu \quad , \quad \log \sigma_u(z) = w_\sigma^\top h_u + c_u^\top z + b_\sigma \quad . \tag{13}$$

Marginalizing over z yields correlated uncertainties. Exact marginalization can be approximated by a small number of samples of z at planning time or by analytical approximations when the dependence is linear-Gaussian.

A Bayesian-ish treatment can be implemented via variational inference over model parameters or via ensembles [27]. Variational inference introduces a distribution $q(\theta)$ and optimizes an evidence lower bound, yielding epistemic

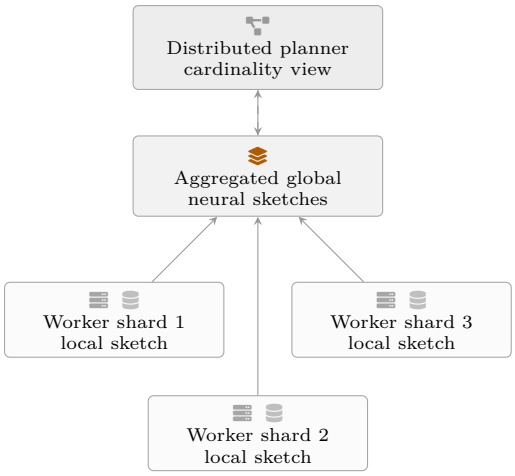

Figure 6: Distributed maintenance of neural sketches across shards. Each worker maintains local probabilistic sketches summarizing data and predicate statistics. These local sketches are periodically merged into an aggregated global view consumed by the planner, allowing cardinality estimation for cross-shard joins without shipping raw data. A lightweight feedback channel updates merging policies and refresh rates based on observed plan quality.

uncertainty. However, full variational training is heavy for planner integration. An alternative is a small ensemble of models trained with different seeds and data resampling, with predictive variance decomposed into epistemic and aleatoric components. The planner can use the combined predictive distribution to hedge against worst-case errors. If $\widehat{Y}^{(e)}$ is the mean prediction of ensemble member e and $\widehat{\sigma}^{(e)}$ its predicted noise, then

$$\widehat{\mu} = \frac{1}{E}\sum_{e=1}^{E} \widehat{Y}^{(e)} \quad , \quad \widehat{\sigma}^2 \approx \frac{1}{E}\sum_{e=1}^{E} \left(\widehat{\sigma}^{(e)}\right)^2 + \frac{1}{E}\sum_{e=1}^{E} \left(\widehat{Y}^{(e)} - \widehat{\mu}\right)^2 \quad . \tag{14}$$

This is computationally manageable when E is small and the inference network is lightweight [28].

Training must accommodate multiple supervision sources. Execution feedback provides observed cardinalities for executed plans, but those plans are chosen based on the current estimator, which creates selection bias. If the system only executes plans that appear good under current estimates, it may under-explore regions where the estimator is wrong. To mitigate this, one can inject controlled exploration in plan selection, such as occasionally evaluating alternative join orders or operator implementations within safe bounds. The estimator is then trained on a more diverse set of observations. Another supervision source is sampling-based selectivity estimation for base relations, which can be collected periodically [29]. Sampling provides unbiased but noisy labels. Combining these sources yields a weighted loss:

$$\mathcal{L}(\theta) = \sum_{t,u} w_{t,u} \cdot \ell\Big(Y_u^{(t)},\ \mu_\theta(u, q^{(t)}),\ \sigma_\theta(u, q^{(t)})\Big) + \beta \cdot \mathcal{R}(\theta) \quad , \tag{15}$$

where $w_{t,u}$ reflects label quality, ℓ is negative log-likelihood, and $\mathcal{R}$ includes regularization and calibration penalties. Calibration can be encouraged by penalizing deviations between empirical coverage and nominal coverage for predicted intervals, implemented via differentiable surrogates.

Backprop-friendly derivatives are essential because sketches and encoders may be trained end-to-end. With Gaussian likelihood, the per-node contribution is [30]

$$\ell(Y, \mu, \sigma) = \frac{1}{2}\log(2\pi) + \log\sigma + \frac{(Y-\mu)^2}{2\sigma^2} \quad , \tag{16}$$

so gradients are

$$\frac{\partial \ell}{\partial \mu} = -\frac{Y-\mu}{\sigma^2} \quad , \quad \frac{\partial \ell}{\partial \sigma} = \frac{1}{\sigma} - \frac{(Y-\mu)^2}{\sigma^3} \quad . \tag{17}$$

If $\sigma = \mathrm{softplus}(\tilde{\sigma}) + \epsilon$, then $\partial\ell/\partial\tilde{\sigma}$ multiplies by $\mathrm{sigmoid}(\tilde{\sigma})$. These derivatives are stable if ϵ prevents σ from approaching zero. For heavy-tailed likelihoods, derivatives remain manageable but require careful numerical implementation.

Approximation error enters through sketch compression, hashing collisions, quantization, and merge operations. A useful abstraction treats the sketch as a noisy observation of an underlying sufficient statistic [31]. Let $s = \bar{s} + \varepsilon$,

where ε captures sketch noise. If the predictor is locally Lipschitz in s, then output error is bounded by $\|\varepsilon\|$. More formally, for mean prediction $\mu_\theta(s)$,

$$|\mu_\theta(s) - \mu_\theta(\bar{s})| \leq L \|s - \bar{s}\|_2 \quad , \tag{18}$$

for some local constant L. This motivates training-time noise injection to reduce L and make predictions robust. For hashing-based sketches, collision bias can be bounded in expectation under pairwise independent hashes, with variance decreasing with bucket count [32]. The neural correction layers learn to reduce residual bias conditioned on predicate features, effectively learning when collisions are likely to matter.

The estimator must also handle approximate queries and partial materializations, such as when using runtime filters or bloom filters, or when adaptive joins change behavior mid-execution. In such cases, the planner needs conditional predictions, such as the selectivity of a filter given that a runtime filter was applied. A principled approach is to condition on additional features that represent these runtime artifacts, such as the parameters of a bloom filter or the observed sample selectivity. This turns the estimator into an online-updatable conditional model, where partial runtime observations refine the posterior over cardinality. A simple update uses Bayesian linear regression on top of neural features, enabling fast posterior updates without retraining the full model [33]. If $\psi_u(q)$ is a neural feature vector for node u, and we model $Y_u \approx \beta^\top \psi_u + \epsilon$ with β having a Gaussian prior, then observing a runtime sample updates the posterior of β in closed form. This provides a lightweight adaptive layer compatible with planning-time constraints.

5 Planner Integration in Distributed Execution

Integrating probabilistic neural sketches into a distributed query planner requires attention to where sketches live, when they are updated, and how inference interacts with plan enumeration and cost modeling. The planner typically maintains metadata services for table statistics, partitioning, and indexes, often stored in a distributed catalog. Neural sketches can be stored alongside these statistics, with per-partition sketches computed during data ingestion, compaction, or periodic maintenance jobs [34]. Because sketches are mergeable, the catalog can store a global sketch per table and optionally per-partition sketches to support skew-aware planning.

At plan time, the planner must estimate cardinalities for many candidate plans. If the join graph has n relations, dynamic programming join enumeration can consider $O(3^n)$ subproblems in the worst case, though practical heuristics reduce this. Even with heuristics, the estimator may be called thousands of times for a single query. Therefore inference must be amortized [35]. A practical strategy caches base relation selectivity predictions keyed by predicate canonicalization and caches intermediate embeddings for partial joins keyed by their relation sets and join predicates. Because neural inference is deterministic given sketches and predicates, memoization reduces repeated computation during enumeration.

The probabilistic output is used in cost models that drive physical operator selection and parallelization. For distributed joins, the planner often chooses between broadcast joins, repartition (shuffle) joins, and hybrid strategies. The choice depends heavily on input sizes and their uncertainty. With distributional estimates, the planner can compute the probability that a broadcast will exceed memory on receivers, or that a shuffle will cause skew-induced stragglers [36]. If S is the estimated size of the build side and M is the per-worker memory budget, then a chance constraint might be

$$\mathbb{P}(S \leq \gamma M) \geq 1 - \delta \quad , \tag{19}$$

where $\gamma < 1$ leaves headroom for overhead. Under a log-normal approximation for size, this probability is computed from μ and σ. The planner can then avoid fragile broadcast plans when uncertainty is high, even if the expected size is small.

Join order optimization under uncertainty can be framed as a robust optimization problem [37]. Let π be a candidate plan and let $\widehat{Y}$ be the predicted mean log-cardinality vector with covariance approximation Σ. If cost is approximately linear in cardinalities in log-space over local ranges, then one can derive a second-order approximation of expected cost. However, distributed operator costs are often piecewise, with thresholds for spilling, repartitioning, and switching algorithms. A practical robust heuristic is to evaluate cost under several quantiles of cardinality, such as median and upper quantile, and choose plans that minimize a weighted combination. This resembles optimizing a weighted sum of Pareto criteria: expected latency and high-quantile latency. The weights are tunable per service-level objective [38].

Query planning itself is a graph algorithm problem. Join graphs can be represented as hypergraphs when join predicates involve multiple attributes. The planner explores join trees or bushy plans, often using dynamic programming with pruning. Because the space is large and optimization is NP-hard, heuristics are common. Neural sketches can inform pruning by providing not only point estimates but also uncertainty that signals when pruning is risky [39]. For example, if two partial plans have similar expected cost but one has much higher uncertainty,

the planner might keep both longer in the search to avoid premature commitment. Conversely, low-uncertainty estimates allow aggressive pruning. This provides a principled method to trade planning time for robustness.

Skew and data locality are central in distributed execution. Cardinality is not just a scalar; distribution across partitions matters. Neural sketches can be extended to produce partition-level selectivity predictions [40]. If per-partition sketches $s_{i,p}$ are available, the estimator can predict $Y_{u,p}$ for the output size in each partition. The planner can then estimate skew metrics, such as the ratio between the largest partition and the mean. For shuffle joins, skew in join keys causes stragglers. If the model predicts per-bucket counts for hashed join keys, it can approximate the maximum load under a given partitioning function. Even a coarse estimate helps choose between hash partitioning schemes or decide whether to apply salting, repartitioning, or adaptive skew-handling operators.

Inference placement matters [41]. In some architectures, the planner runs on a coordinator node with limited compute. Neural inference can be done on the coordinator if the model is small and sketches are cached locally. Alternatively, inference can be offloaded to a metadata service with accelerators or to worker nodes. Offloading introduces network latency and serialization overhead. The neural sketch design reduces this by using small vectors and enabling the coordinator to compute estimates using only sketches and predicate encodings [42]. For large clusters, sketches can be stored in a distributed cache with local replicas. Communication efficiency is influenced by sketch dimension k, quantization bits q, and update frequency. If sketches are updated periodically, the amortized bandwidth per second is roughly proportional to $(kq) \times$ update rate. A planner-level budget can be enforced by choosing k and q and by updating only when drift is detected.

Drift detection can be implemented by comparing current sketch summaries to prior ones using a distance metric. If $s^{(t)}$ is the sketch at time t, a normalized cosine distance or Mahalanobis distance in a learned embedding space can indicate distribution shifts. For example,

$$d(t, t') = 1 - \frac{\langle s^{(t)}, s^{(t')} \rangle}{\|s^{(t)}\|_2 \, \|s^{(t')}\|_2} \quad , \tag{20}$$

and the system triggers refresh if d exceeds a threshold [43]. More principled detection uses a learned projection P and tracks $\|Ps^{(t)} - Ps^{(t')}\|_2$ to emphasize workload-relevant directions.

A final integration aspect is feedback capture. Execution engines can report actual cardinalities at runtime, often already available for adaptive execution or monitoring. Collecting these for training requires careful sampling to avoid overhead. The system can log only a subset of nodes, such as the most expensive operators or operators with high uncertainty. The feedback pipeline must anonymize or canonicalize predicates if needed, and must align logged nodes to plan features used by the estimator [44]. Feedback is then used for periodic retraining or incremental updates. Because planner behavior depends on the estimator, updates should be staged to avoid sudden plan shifts; one approach is to blend new and old models in an ensemble during a transition period and to monitor regression in plan quality metrics.

6 Complexity, Storage Internals, and Performance Engineering

The computational complexity of the estimator matters at two timescales: maintenance-time sketch computation and planning-time inference. Maintenance-time cost is dominated by scanning tuples to update sketches. If each tuple contributes $O(k)$ operations via a linear map Wx, the per-tuple cost is $O(kd_x)$, which may be too expensive if implemented naively [45]. Practical implementations exploit sparsity in x via hashing and sparse matrix multiplication. If each tuple activates s nonzero features, the cost becomes $O(ks)$. Additionally, W can be structured, such as using a product of low-rank matrices or fast transforms, reducing cost. For example, if $W = UV^\top$ with rank $r \ll \min(k, d_x)$, then computing Wx costs $O(rd_x + rk)$, and with sparsity it becomes $O(rs + rk)$. This connects sketch computation to low-rank approximation and performance engineering [46].

Planning-time inference must handle many candidate plans. If the estimator is an MLP with hidden dimension h and input dimension roughly $k + d$, a single forward pass costs $O(h(k + d) + h^2)$ for dense layers. With caching, base node inference is done once per referenced relation-predicate pair. Join node inference is done per candidate join, and the number of joins considered depends on enumeration strategy. In dynamic programming for left-deep trees, the number of join evaluations is roughly $O(n2^n)$ in the worst case, but practical planners cap n or use greedy heuristics. Therefore inference must be efficient enough that it does not dominate planning time [47]. In deployed settings, inference budgets may be on the order of milliseconds, so model size is constrained. Quantized inference and vectorization can help, and the model can be compiled or fused to reduce overhead.

Storage internals influence what features are available for sketches and how cheaply they can be computed. Columnar storage formats, such as those with dictionary encoding, run-length encoding, and zone maps, already maintain metadata that can be reused. For example, per-row-group min/max statistics can provide coarse bounds for range predicates [48]. Neural sketches can incorporate these as auxiliary inputs to improve robustness under compressed storage. When dictionary encoding is present, maintaining approximate frequency information is

natural, and sketch updates can piggyback on dictionary maintenance. For nested data, storing path-level statistics can be expensive; hashed path signatures provide a compact alternative suitable for sketches.

Data structure choices affect both sketch storage and access. Sketch vectors for many tables and partitions can be stored in a contiguous memory region to improve cache locality [49]. If the planner frequently requests sketches for a subset of hot tables, an LRU cache keyed by table id can reduce catalog lookups. Because sketches are mergeable, the system can store a hierarchy of sketches, such as per-partition, per-node, and global, enabling locality-aware planning. For example, if a query is constrained to a specific partition range, the planner can merge only relevant sketches rather than using the global one, improving accuracy for partition-pruned queries.

Communication efficiency can be analyzed through a coding lens. Suppose sketches are transmitted from workers to the coordinator during maintenance [50]. If sketches are quantized to q bits per component and dimension is k, then the raw payload is kq bits. If sketches have correlated components, entropy coding can reduce this. However, entropy coding introduces CPU overhead and complicates random access. A practical compromise uses block-floating quantization with per-block scales, reducing effective entropy while keeping decoding cheap. The relationship between quantization error and estimator accuracy can be studied by modeling the sketch as a random vector with covariance Σ_s and quantization noise as additive uniform noise. The expected squared error per component scales with the quantization step size squared, and total noise energy scales with k [51]. This yields a trade-off between bandwidth and estimation quality, which can be optimized under a communication budget using a Lagrangian:

$$\min_{k,q} \quad \mathbb{E}[\text{Loss}(k,q)] \quad \text{subject to} \quad kq \le B \quad , \tag{21}$$

$$\min_{k,q} \quad \mathbb{E}[\text{Loss}(k,q)] + \lambda \cdot (kq - B) \quad , \tag{22}$$

where B is the bit budget. While k and q are discrete, heuristics can search over feasible pairs.

Error analysis must separate sources. For base table selectivity, errors arise from limited sketch capacity to encode joint distributions and from predicate encoding ambiguity [52]. For joins, errors arise additionally from collision bias in hashed key sketches and from conditioning mismatch when upstream filters change key distributions. An approximate bound for join cardinality estimation using hashed sketches can be motivated by variance of a dot-product estimator. If f and g are frequency vectors and we estimate their dot product via a Count-Sketch-like mechanism, the estimator is unbiased under certain assumptions and its variance decreases with bucket count B. Neural correction layers can reduce mean-squared error further, but they introduce model bias. The overall error can be decomposed into approximation bias and variance due to sketching and data noise [53]. In probabilistic terms, aleatoric uncertainty captures irreducible variability due to finite observation and hashing, while epistemic uncertainty captures model uncertainty due to limited training data and workload shift.

Planner search itself has complexity concerns. Join order optimization by dynamic programming runs in time exponential in n in the worst case, and distributed planning often adds considerations for exchanges and repartitioning, which multiply choices. This is a classic NP-hard space, and learned components do not change the worst-case complexity. What neural sketches can do is improve pruning and guide heuristics. A heuristic can be framed as approximating an optimal plan under a learned cost-to-go function [54]. However, cost-to-go learning is risky if it reduces exploration too much. A safer approach uses learned estimates as one signal among others, with fallback rules ensuring correctness and bounded resource usage. For example, when uncertainty is high, the planner can choose adaptive join operators that can switch build/probe sides or spill gracefully, trading slightly higher expected latency for robustness.

Gradient descent variants matter for training stability, especially under feedback-driven data that is nonstationary. Adaptive optimizers such as Adam or RMSProp can converge quickly but may be sensitive to changing label distributions [55]. A schedule that mixes adaptive steps with occasional second-order preconditioning can help, but must remain lightweight. Because sketches and encoders may be updated over time, catastrophic forgetting is a risk. Regularization can include anchoring parameters to previous values or using replay buffers of older queries. From a systems perspective, training should be decoupled from serving; models are trained offline or in a separate service, then deployed with versioning and rollback. Reproducibility requires deterministic preprocessing of predicates, fixed random seeds for hash functions, and logged model versions tied to query logs [56].

Performance engineering also includes inference batching. During planning, many candidate nodes share the same relation sketch and differ only in predicate constants. The planner can batch predicate encodings and run vectorized inference, reducing overhead. Similarly, join inference can batch candidate joins at the same enumeration stage. If the model is implemented in a JIT-compiled runtime or a specialized inference engine, these batches can be fused [57]. However, batching must not increase planning latency unpredictably. A bounded batching strategy uses micro-batches within a fixed time slice, ensuring responsive planning.

7 Evaluation Protocol and Discussion of Results

Evaluating probabilistic cardinality estimation for complex predicates requires metrics beyond mean absolute error on a static dataset. The primary objective is to improve planner decisions, which should be measured by end-to-end runtime, resource consumption, and plan stability under workload variability. A robust evaluation protocol separates training, validation, and test workloads by time and by predicate patterns to test generalization. It also includes shift tests where data distributions change, such as after ingesting new time partitions or after application feature changes that alter predicate mix [58]. Because distributed execution introduces noise from cluster variability, repeated runs are needed to estimate statistical uncertainty in observed runtimes, and the evaluation should report variability rather than only point averages.

Accuracy metrics should reflect the log-scale nature of cardinalities and the planner's sensitivity. A common metric is mean absolute error in $Y = \log(N + 1)$, which corresponds to multiplicative error in N. However, planner decisions are often dominated by threshold effects, such as whether to broadcast or shuffle, whether to spill, or whether to choose a hash join versus a sort-merge join. Therefore, classification-style metrics can be used for decision thresholds, measuring how often the estimator correctly predicts which side of a threshold the true cardinality lies [59]. Probabilistic predictions enable calibration metrics: for predicted intervals at nominal coverage, the empirical coverage should match. Calibration can be evaluated by grouping predictions by predicted variance and measuring residual distribution. Underestimation of uncertainty is particularly harmful because it leads to overconfident fragile plans.

Planner-level evaluation should compare several configurations: a baseline planner with classical statistics, a planner with neural sketches but deterministic point estimates, and a planner with probabilistic outputs used in risk-aware planning. The comparison should hold planning budgets constant, including time spent on estimation and enumeration [60]. If the neural estimator increases planning time, it must be accounted for because planning latency is a user-visible cost. In many interactive systems, planning time must remain small relative to execution time. A system-level metric is time-to-first-row or time-to-first-result, which can be sensitive to planning overhead.

Complex predicate workloads should include conjunctions of correlated filters, disjunctions, predicate pushdown through projections, and user-defined functions. For user-defined functions, ground truth selectivity can be hard to predict from base statistics. The evaluation can treat UDF identifiers and coarse input features as part of the predicate encoding and measure how well the model generalizes across UDF invocations [61]. Semi-structured predicates are evaluated by including path expressions and existence checks. Because schema evolution can change paths over time, the model should be tested under missing or new paths, where predicate encoding must fall back gracefully.

Distributed execution mechanics should be explicitly included. For example, for shuffle joins, the evaluation should measure skew-induced stragglers and the effect of estimates on partitioning choices. If the model predicts per-partition cardinalities, the planner can choose skew-handling strategies, and the evaluation can measure reduction in tail latency [62]. Tail metrics such as p95 and p99 runtime are often more relevant than mean, especially in multi-tenant settings. Probabilistic planning aims to reduce tail risk by avoiding plans that are optimal only under optimistic estimates. Therefore, a key outcome is a shift in the distribution of runtimes toward lower variance, even if mean improvements are modest.

Reproducibility requires controlling sources of nondeterminism. Hash functions used in sketches must be fixed and logged [63]. Sketch maintenance jobs must be deterministic given input data ordering, or must use commutative aggregation that makes ordering irrelevant. Training pipelines must log preprocessing, model architectures, hyperparameters, and random seeds. Execution environments should pin cluster size and configuration where possible, and should record system load. Because distributed systems are noisy, reproducibility also requires reporting confidence intervals or at least variance across repeated trials. When feedback-driven training is used, the sequence of model updates influences the observed data, so evaluation should separate offline estimation accuracy from online co-adaptation effects [64]. A practical reproducible setup uses a logged workload trace and simulates planning decisions with fixed model snapshots, then validates selected plans in a controlled cluster.

Discussion of results should emphasize failure modes as much as successes. Neural sketches may struggle when predicates involve rare values unseen during training, when schema changes invalidate predicate encodings, or when join keys have adversarial collision patterns under hashing. Quantization can harm accuracy if sketch components have heavy-tailed distributions; clipping and robust normalization mitigate this but may lose signal. Another failure mode occurs when feedback is biased toward certain plan shapes, leading the estimator to underperform on unexecuted plan regions. Controlled exploration helps but must be bounded by safety constraints [65]. The probabilistic model can also be miscalibrated under drift; ensemble variance may underestimate uncertainty if all models share the same blind spots. Monitoring calibration online and triggering retraining is therefore part of the operational story.

A useful way to interpret improvements is through the lens of plan stability. Traditional estimators can cause plan oscillations: small changes in predicate constants lead to large changes in estimated cardinality and thus to different join orders, which can change runtime dramatically. Neural sketches, when trained with smoothness

regularization and robust features, can reduce these discontinuities [66]. Probabilistic planning further reduces oscillations by preferring plans whose expected performance remains acceptable under a range of cardinalities. This is aligned with multi-objective optimization, where minimizing worst-case or high-quantile latency is a meaningful objective alongside expected latency. From a systems perspective, such stability can reduce cache churn, improve predictability of resource scheduling, and reduce the incidence of emergency spill or replan events.

8 Conclusion

This paper presented a probabilistic framework for cardinality estimation under complex predicates in distributed query planners, centered on neural sketches that compress relation and join-relevant information into mergeable embeddings. By treating cardinality as a random variable conditioned on predicate structure, query graph context, and compact data summaries, the approach supports both accurate point estimates and calibrated uncertainty, enabling risk-aware planning [67]. The design emphasized distributed constraints: sketches are streaming-computable, mergeable across partitions, and compressible under tight communication budgets; inference is lightweight and cacheable to support repeated evaluation during plan enumeration. The modeling incorporated representation learning for predicate semantics, similarity-style interactions between predicate embeddings and relation sketches, and uncertainty mechanisms that can be implemented via heteroscedastic likelihoods and small ensembles. Systems integration addressed inference placement, drift detection, feedback capture, and robustness to skew and threshold effects common in distributed execution.

The analysis connected sketch capacity and quantization to approximation error, used low-rank and projection viewpoints to motivate correlation capture, and framed planner decisions under uncertainty via multi-objective and constrained optimization. Evaluation considerations focused on planner-level outcomes, tail latency, plan stability, and reproducibility under distributed noise. Overall, probabilistic neural sketches provide a structured route to improving decision quality for complex predicates without requiring heavyweight per-query sampling or brittle independence assumptions, while still respecting the operational realities of distributed query optimization [68].

References

[1] "Scalable distributed systems," in Springer New York, Jun. 12, 2018, pp. 2290–2290. DOI: 10.1007/978-1-4939-7131-2_101017

[2] K. A. Jothy, K. Sivakumar, and M. J. Delsey, "Distributed system framework for mobile cloud computing," *Bonfring International Journal of Research in Communication Engineering*, vol. 8, no. 1, pp. 05–09, Feb. 28, 2018. DOI: 10.9756/bijrce.8357

[3] C. Bicer, I. Murturi, P. K. Donta, and S. Dustdar, "Blockchain-based zero trust on the edge," in *2023 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, Dec. 13, 2023, pp. 1006–1013. DOI: 10.1109/csci62032.2023.00167

[4] T. M. Tatarnikova and E. M. Arkhiptsev, "Hybrid time synchronization in distributed systems," in *2025 XXVIII International Conference on Soft Computing and Measurements (SCM)*, IEEE, May 28, 2025, pp. 307–310. DOI: 10.1109/scm66446.2025.11060106

[5] H. E. Fazazi, M. Elgarej, M. Qbadou, and K. Mansouri, "Design of an adaptive e-learning system based on multi-agent approach and reinforcement learning," *Engineering, Technology & Applied Science Research*, vol. 11, no. 1, pp. 6637–6644, Feb. 6, 2021. DOI: 10.48084/etasr.3905

[6] R. Malik et al., "Mlr-index: An index structure for fast and scalable similarity search in high dimensions," in *International Conference on Scientific and Statistical Database Management*, Springer, 2009, pp. 167–184.

[7] A. Sallam, N. Aklan, N. Aklan, and T. H. Rassem, "A dense memory representation using bitmap data structure for improving ndn push-traffic model," *Annals of Telecommunications*, vol. 79, no. 1-2, pp. 73–83, Jul. 24, 2023. DOI: 10.1007/s12243-023-00972-9

[8] W. B. Daszczuk, "Modeling and verification of asynchronous systems using timed integrated model of distributed systems.," *Sensors (Basel, Switzerland)*, vol. 22, no. 3, pp. 1157–1157, Feb. 3, 2022. DOI: 10.3390/s22031157

[9] A. M. Ahmed, M. A. Saeed, A. A. Hamood, A. A. Alazab, and K. A. Ahmed, "Comparative study of static analysis and machine learning approaches for detecting android banking malware," in *2023 3rd International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, IEEE, Oct. 10, 2023, pp. 1–8. DOI: 10.1109/esmarta59349.2023.10293602

[10] Y. Li, Y. Liu, D. Zou, and H. Wang, "Ensemble pruning via graph neural networks," in *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, ACM, Nov. 10, 2025, pp. 1716–1725. DOI: 10.1145/3746252.3761349

[11] R. Chotkan, J. Decouchant, and J. Pouwelse, "Distributed attestation revocation in self-sovereign identity," in *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, vol. 2014, IEEE, Sep. 26, 2022, pp. 414–421. DOI: 10.1109/lcn53696.2022.9843323

[12] D. Vasilas, *Search on secondary attributes in geo-distributed systems*, Jan. 1, 2018. DOI: 10.48550/arxiv.1801.02974

[13] X. Gao et al., "Private set intersection-based ad click detection," in *2025 11th IEEE International Conference on Privacy Computing and Data Security (PCDS)*, IEEE, Aug. 1, 2025, pp. 330–336. DOI: 10.1109/pcds65695.2025.00052

[14] E. Becks, M. Josten, V. Matkovic, and T. Weis, "Revising poor man's eye tracker for crowd-sourced studies," in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, IEEE, Mar. 13, 2023, pp. 328–330. DOI: 10.1109/percomworkshops56833.2023.10150386

[15] D. Marek, P. Biernacki, J. Szygula, and A. Domanski, "General concepts of a simulation method for automated guided vehicle in industry 4.0," in *2022 IEEE International Conference on Big Data (Big Data)*, IEEE, Dec. 17, 2022, pp. 6306–6314. DOI: 10.1109/bigdata55660.2022.10020846

[16] J. Hecking-Harbusch and N. O. Metzger, "Atva - efficient trace encodings of bounded synthesis for asynchronous distributed systems," in Germany: Springer International Publishing, Oct. 21, 2019, pp. 369–386. DOI: 10.1007/978-3-030-31784-3_22

[17] D. Efimov, A. Polyakov, and A. Aleksandrov, *Proofs for "discretization of homogeneous systems using euler method with a state-dependent step"*, Jun. 24, 2019.

[18] H. H. Ali and A. D. S. H. Shaker, "Techniques for secure distributed systems," *Journal of Physics: Conference Series*, vol. 1530, no. 1, pp. 012 006–, May 1, 2020. DOI: 10.1088/1742-6596/1530/1/012006

[19] R. Buyya and R. T. Barry, "Speaker," in *2022 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*, IEEE, Sep. 16, 2022, pp. 1–3. DOI: 10.1109/icbds53701.2022.9936030

[20] 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*, IEEE, 2006, pp. 628–629.

[21] M. Krogh-Jespersen, A. Timany, M. E. Ohlenbusch, S. O. Gregersen, and L. Birkedal, "Esop - aneris: A mechanised logic for modular reasoning about distributed systems," in Germany: Springer International Publishing, Apr. 18, 2020, pp. 336–365. DOI: 10.1007/978-3-030-44914-8_13

[22] P. Raith, S. Nastic, and S. Dustdar, "Simuscale: Optimizing parameters for autoscaling of serverless edge functions through co-simulation," in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, IEEE, Jul. 7, 2024, pp. 305–315. DOI: 10.1109/cloud62652.2024.00042

[23] P. Ocenasek, "Effective synchronization of data in distributed systems," in CRC Press, Jun. 12, 2018, pp. 177–180. DOI: 10.1201/b18658-38

[24] D. Vasilas, *Search on secondary attributes in geo-distributed systems.* Jan. 9, 2018.

[25] J. Pennekamp, R. Matzutt, S. S. Kanhere, J. Hiller, and K. Wehrle, "The road to accountable and dependable manufacturing," *Automation*, vol. 2, no. 3, pp. 202–219, Sep. 13, 2021. DOI: 10.3390/automation2030013

[26] E. Azketa, X. Mendialdua, I. Ibarguren, and A. Solís, "Método de sincronización para sistemas distribuidos con seguridad funcional," *Revista Iberoamericana de Automática e Informática industrial*, vol. 18, no. 2, pp. 113–118, Apr. 6, 2021. DOI: 10.4995/riai.2020.14022

[27] Z. Mayransaev and A. B. Chernyshev, "Generalized circular criterion for the absolutesustainability of distributed systems," *IZVESTIYA SFedU. ENGINEERING SCIENCES*, no. 2, pp. 182–189, Jul. 1, 2021. DOI: 10.18522/2311-3103-2021-2-182-189

[28] A. S. Hosseini and S. Staab, "Emotional framing in the spreading of false and true claims," in *Proceedings of the 15th ACM Web Science Conference 2023*, ACM, Apr. 30, 2023, pp. 96–106. DOI: 10.1145/3578503.3583611

[29] A. Poshtkohi and M. B. Ghaznavi-Ghoushchi, "Iot and distributed systems," in Auerbach Publications, Feb. 9, 2023, pp. 15–22. DOI: 10.1201/9781003379041-2

[30] S. Wang et al., "Vg-net: Sensor time series anomaly detection with joint variational autoencoder and graph neural network," in *2024 IEEE Smart World Congress (SWC)*, IEEE, Dec. 2, 2024, pp. 456–463. DOI: 10.1109/swc62898.2024.00094

[31] D. Woos, Z. Tatlock, M. D. Ernst, and T. Anderson, *A graphical interactive debugger for distributed systems.* Jun. 13, 2018.

[32] F. Loisel, G. Zeqo, A. Morichetta, A. Lackinger, and S. Dustdar, "Raincloud: Decentralized coordination and communication in heterogeneous iot swarms," in *2024 International Symposium on Parallel Computing and Distributed Systems (PCDS)*, IEEE, Sep. 21, 2024, pp. 1–10. DOI: 10.1109/pcds61776.2024.10743766

[33] A. Charapko, A. Ailijiang, M. Demirbas, and S. S. Kulkarni, "Retroscope: Retrospective monitoring of distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 11, pp. 2582–2594, Nov. 1, 2019. DOI: 10.1109/tpds.2019.2911944

[34] N. Phu, *Clara algorithm in a distributed system*, Sep. 5, 2018.

[35] 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*, 2007, pp. 2701–2706.

[36] A. Almen and D. Dentcheva, "Fair risk optimization of distributed systems," *Annals of Operations Research*, Nov. 5, 2025. DOI: 10.1007/s10479-025-06917-w

[37] N. Evangelou-Oost, C. Bannister, and I. J. Hayes, *Contextuality in distributed systems*, Jan. 1, 2022. DOI: 10.48550/arxiv.2210.09476

[38] V. D. Maio, A. Aral, and I. Brandic, "A roadmap to post-moore era for distributed systems," in *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, ACM, Jul. 25, 2022, pp. 30–34. DOI: 10.1145/3524053.3542747

[39] S. Kamara, "Encrypted distributed systems," in *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, ACM, Jul. 20, 2022, pp. 163–163. DOI: 10.1145/3519270.3538454

[40] S. H. S. A. UBAIDILLAH, N. AHMAD, and N. A. Sahabudin, "A survey on potential reactive fault tolerance approach for distributed systems in big data," in *Third International Conference on Computer Vision and Information Technology (CVIT 2022)*, SPIE, Feb. 24, 2023, pp. 21–21. DOI: 10.1117/12.2670017

[41] T. Pusztai, S. Nastic, P. Raith, S. Dustdar, D. Vij, and Y. Xiong, "Vela: A 3-phase distributed scheduler for the edge-cloud continuum," in *2023 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, Sep. 25, 2023, pp. 161–172. DOI: 10.1109/ic2e59103.2023.00026

[42] N. V. Mokrova, A. M. Mokrov, and A. V. Safonova, "The distributed systems of engineering supervision of permanent structures," *IOP Conference Series: Materials Science and Engineering*, vol. 365, no. 6, pp. 062 036–, Jun. 12, 2018. DOI: 10.1088/1757-899x/365/6/062036

[43] A. Lapkovskis, B. Sedlak, S. Magnússon, S. Dustdar, and P. K. Donta, "Benchmarking dynamic slo compliance in distributed computing continuum systems," in *2025 IEEE International Conference on Edge Computing and Communications (EDGE)*, IEEE, Jul. 7, 2025, pp. 93–102. DOI: 10.1109/edge67623.2025.00020

[44] "Clock synchronization in distributed systems," in CRC Press, Oct. 3, 2018, pp. 281–292. DOI: 10.1201/9781315218434-26

[45] I. Schagaev, "Distributed systems: Maximizing resilience," in Springer International Publishing, Jul. 10, 2019, pp. 249–266. DOI: 10.1007/978-3-030-21244-5_18

[46] W. Yin, B. B. Zhu, Y. Xie, P. Zhou, and D. Feng, "Backdoor attacks on bimodal salient object detection with rgb-thermal data," in *Proceedings of the 32nd ACM International Conference on Multimedia*, ACM, Oct. 28, 2024, pp. 4110–4119. DOI: 10.1145/3664647.3681096

[47] *ACSD - Concurrent Secrets with Quantified Suspicion*, Apr. 4, 2018.

[48] N. Naik, "Isse - demystifying properties of distributed systems," in *2021 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, Sep. 13, 2021, pp. 1–8. DOI: 10.1109/isse51541.2021.9582515

[49] M. Aiello, *The Web Was Done by Amateurs: A Reflection on One of the Largest Collective Systems Ever Engineered.* Jul. 20, 2018.

[50] 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*, IEEE, 2006, pp. 1–6.

[51] G. Chourdakis et al., "Precice v2: A sustainable and user-friendly coupling library.," *Open research Europe*, vol. 2, pp. 51–51, Sep. 30, 2022. DOI: 10.12688/openreseurope.14445.2

[52] *Distributed systems*, Dec. 2, 2020. DOI: 10.1002/9781119644682.ch7

[53] W. A. Mohammad et al., "A survey of data mining activities in distributed systems," *Asian Journal of Research in Computer Science*, pp. 1–18, Sep. 7, 2021. DOI: 10.9734/ajrcos/2021/v11i430267

[54] M. Zaccarini et al., "Telka: Twin-enhanced learning for kubernetes applications," in *2024 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, Jun. 26, 2024, pp. 1–6. DOI: 10.1109/iscc61673.2024.10733736

[55] S. Joshi, S. Ameta, and G. Lavania, "Balanced load in distributed system with nosql middleware," *Journal of emerging technologies and innovative research*, May 1, 2019.

[56] A. A. Klishin, D. J. Singer, and G. van Anders, "Avoidance, adjacency, and association in distributed systems design," *Journal of Physics: Complexity*, vol. 2, no. 2, pp. 025 015–, Apr. 8, 2021. DOI: 10.1088/2632-072x/abe27f

[57] 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)*, IEEE, 2006, pp. 44–44.

[58] T. Heins et al., "Delay-aware model predictive control for fast frequency control," in *2023 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, IEEE, Oct. 31, 2023, pp. 1–7. DOI: 10.1109/smartgridcomm57358.2023.10333921

[59] *OSDI - Scalable Memory Protection in the PENGLAI Enclave*, Jun. 23, 2021.

[60] R. Meng, G. Pîrlea, A. Roychoudhury, and I. Sergey, "Greybox fuzzing of distributed systems," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ACM, Nov. 15, 2023, pp. 1615–1629. DOI: 10.1145/3576915.3623097

[61] T. D. M. Brizuela, D. L. L. R. Martínez, F. Agostini, and J. T. F. Martínez, "Intelligent process migration in heterogeneous distributed systems," *Computing and Artificial Intelligence*, pp. 2018–2018, Dec. 18, 2024. DOI: 10.59400/cai2018

[62] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar, "Diffusing high-level slo in microservice pipelines," in *2024 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, Jul. 15, 2024, pp. 11–19. DOI: 10.1109/sose62363.2024.00008

[63] V. Kale, "Distributed systems," in CRC Press, Jul. 8, 2019, pp. 71–94. DOI: 10.1201/9781351029148-5

[64] Z. Dong et al., "Fine-grained re-execution for efficient batched commit of distributed transactions," *Proceedings of the VLDB Endowment*, vol. 16, no. 8, pp. 1930–1943, Jun. 22, 2023. DOI: 10.14778/3594512.3594523

[65] 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*, 2009, p. 18.

[66] R. Meng, G. Pîrlea, A. Roychoudhury, and I. Sergey, *Greybox fuzzing of distributed systems*, Jan. 1, 2023. DOI: 10.48550/arxiv.2305.02601

[67] G. Farina, *Tractable reliable communication in compromised networks*, Dec. 21, 2020.

[68] 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)*, IEEE, Aug. 5, 2025, pp. 1–9. DOI: 10.1109/esmarta66764.2025.11132254