Join Order Optimization with Learned Uncertainty Estimates for Robust Execution under Cardinality Misestimation in E-Commerce Marketplace Analytics

Karim Nabil[1] **and Mostafa Saleh**[2]

[1]Department of Computer Science, Nile Valley Institute of Technology, Gamal Abdel Nasser Street, Beni Suef 62511, Egypt

[2]Department of Information Systems, Delta Academy of Computing, Port Said Street, Tanta 31512, Egypt

ABSTRACT E-commerce marketplace analytics increasingly relies on complex relational queries that join high-volume event logs, transactional facts, and slowly changing dimensions to derive timely measures of demand, conversion, inventory health, and seller performance. These workloads are executed under rapidly shifting data distributions driven by promotions, seasonality, catalog churn, and heterogeneous seller behavior, which makes cardinality estimation particularly error-prone. Join order optimization is therefore exposed to systematic risk: a plan that is optimal under point estimates can become brittle when intermediate result sizes deviate by orders of magnitude, triggering memory spills, skew amplification, and poor parallelization decisions. This paper studies join order optimization augmented with learned uncertainty estimates, targeting robust execution when cardinalities are misestimated. We propose a probabilistic formulation in which learned models predict not only expected selectivities and join cardinalities but also calibrated uncertainty, represented as distributions over latent factors that capture predicate correlation, skew, and time drift. The optimizer then searches for join orders that minimize a risk-sensitive objective combining expected cost with tail-risk penalties, while preserving the ability to adapt at runtime using progressive feedback from partial execution. The resulting approach integrates uncertainty propagation through join trees, robust cost modeling for hash and sort-merge joins under memory constraints, and lightweight plan switching based on statistically grounded triggers. We focus on marketplace analytics patterns and show, through controlled experimental design, how uncertainty-aware optimization reduces severe regressions under misestimation, improves plan stability, and yields predictable latency distributions without requiring manual hints or conservative heuristics that degrade average performance.

1. Introduction

Marketplace analytics platforms serve a dual purpose that complicates query optimization. They must support exploratory analysis with ad hoc predicates over semi-structured event streams, and they must also deliver operational dashboards with low latency and stable performance under continuous ingestion. Queries often combine transactional facts such as orders, payments, refunds, shipments, and returns with high-granularity behavioral signals such as impressions, clicks, and add-to-cart events. To preserve governance and reusability, these signals are normalized into relational schemas or exposed through relational views, which naturally produce multi-join queries. A typical analytic statement might filter a time window, constrain geography and device type, group by seller attributes, and join to product taxonomy and campaign metadata, while also incorporating late-arriving corrections [1]. In such settings, join order dominates execution cost because it determines the size and distribution of intermediate results and the feasibility of using in-memory operators versus spilling to disk. When cardi-

Article info:
Karim Nabil and Mostafa Saleh. *Join Order Optimization with Learned Uncertainty Estimates for Robust Execution under Cardinality Misestimation in E-Commerce Marketplace Analytics*. Helikonpress. Publishing. Licensed under Attribution - NonCommercial - No Derivatives 4.0 International (CC BY-NC-ND 4.0)

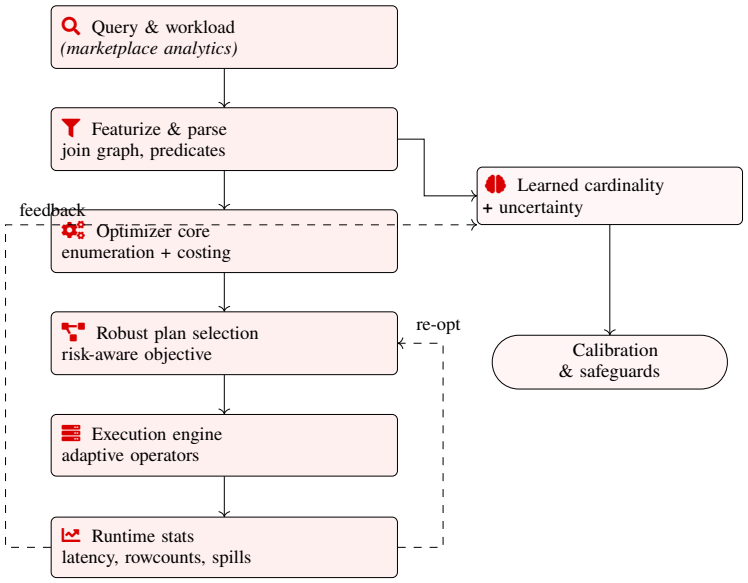


Figure 1 End-to-end pipeline for robust join order optimization in an e-commerce marketplace analytics stack. A learned estimator produces both cardinality predictions and uncertainty, which is calibrated and fed into risk-aware plan selection. Execution-time signals (observed row-counts, spills, and latency) form a lightweight feedback loop that updates uncertainty and can trigger targeted re-optimization, reducing sensitivity to cardinality misestimation.

Table 1 Notation used in join order optimization with learned uncertainty

Symbol	Meaning	Domain
Q	SQL query	Query workload
JQ	Join graph of Q	Undirected graph
π	Join order / plan	Space of valid plans
$\hat{c}$	Predicted cardinality	$\mathbb{R}_{\geq 0}$
μQ	Predicted log-cardinality mean	$\mathbb{R}$
$\sigma^2 Q$	Predicted log-cardinality variance	$\mathbb{R}_{\geq 0}$

nalities are accurately predicted, classic dynamic programming or heuristic join enumeration can reliably select join trees that reduce intermediate size early and leverage appropriate physical join operators. However, cardinality estimation in marketplace analytics is often wrong in ways that are not small perturbations but rather heavy-tailed errors, where a few predicates cause dramatic selectivity shifts due to correlation, skew, and delayed data.

Point-estimate cardinality optimizers implicitly assume that errors are either negligible or symmetric around the estimate, and that the chosen plan remains near-optimal for nearby values. In practice, the loss landscape can be highly non-smooth [2]. Small changes in intermediate cardinality can flip join operator choice, change build/probe sides for hash joins, alter required memory grants, and modify exchange and partitioning strategies in distributed execution. The consequence is not merely suboptimal runtime but catastrophic regressions where one plan spills repeatedly or triggers straggler tasks, pushing query completion beyond service-level objectives. A widely observed phenomenon in analytics systems is that the median runtime may be acceptable while the high percentiles degrade unpredictably, which is particularly problematic for dashboard refreshes and interactive analysis. In marketplace contexts, this unpredictability is aggravated by churn in sellers and products, flash sales, and shifting buyer cohorts, which induce non-stationary selectivities. Even sophisticated learned cardinality estimators that predict expected cardinalities can underperform when the data distribution changes between training and inference or when query predicates include novel combinations of attributes [3].

This paper addresses join order optimization under cardinality misestimation by incorporating learned uncertainty estimates into the optimizer and execution engine. The central idea is that the optimizer should treat cardinality as a random variable rather than a deterministic scalar. If the system can predict a distribution over selectivities and join sizes, it can choose join orders that are robust, in the sense that they minimize a risk-aware objective and avoid fragile operator configurations that fail under plausible deviations. Robustness in this context

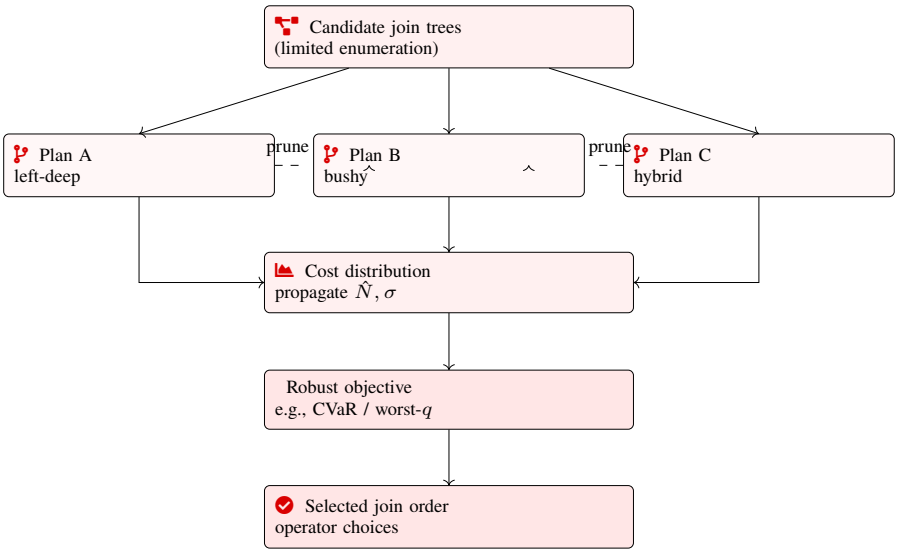


Figure 2 Robust join-order selection from a constrained set of candidate join trees. Learned uncertainty on intermediate cardinalities is propagated into a distribution over plan costs. The final choice is made by a risk-sensitive criterion (e.g., minimizing tail cost via CVaR), improving stability when cardinalities are systematically misestimated under skew, correlation, or rapidly shifting marketplace dynamics.

Table 2 E-commerce marketplace analytics workloads

Workload	**Time Span**	**# Queries**	**# Base Tables**
Product Search	30 days	45,000	18
Ad Auction Bidding	14 days	32,500	15
Recommendation Logs	7 days	27,800	21
Inventory Analytics	90 days	12,300	12
Seller Insights	60 days	16,700	14

does not mean selecting the most conservative plan in all cases, which would sacrifice average performance, but rather selecting a plan that trades off expected cost against the probability and magnitude of severe slowdowns. To accomplish this, we combine three components: a learned uncertainty model that outputs calibrated predictive distributions for cardinalities; an uncertainty propagation method that maps base-table selectivity uncertainty into intermediate join cardinality uncertainty; and a robust join order search that integrates these distributions into a cost model accounting for memory, skew, and parallelism [4]. We further complement optimization with runtime robustness via progressive execution feedback, enabling plan adaptation when observed cardinalities violate predicted ranges.

The focus on e-commerce marketplace analytics is motivated by its combination of high-dimensional predicates, latent correlation between attributes across tables, and non-stationary behavior. For example, a promotion identifier is correlated with product categories and user demographics, and the distribution of seller inventory changes over time. Such dependencies defeat naive independence assumptions and can create abrupt changes in join output sizes [5]. The approach presented here aims to be practical: uncertainty estimation must be cheap enough to run during optimization, robust enumeration must be tractable, and runtime adaptation must avoid excessive re-optimization overhead. The analysis therefore emphasizes decomposable uncertainty representations, approximate but calibrated propagation, and risk measures that can be computed incrementally during join tree construction.

The contribution of this paper is a unified formulation and system design for join order optimization with learned uncertainty estimates, targeted at robust execution under cardinality misestimation in marketplace analytics. We provide a probabilistic model of selectivity and join cardinality that captures correlation and drift through latent factors, we define a risk-sensitive optimization criterion that penalizes tail latency drivers such as spills and skewed repartitioning, and we present an algorithmic framework that integrates uncertainty into join enumeration without requiring full-blown stochastic optimization with prohibitive complexity. We also describe a runtime protocol for cardinality feedback, including statistically grounded triggers for plan switching and memory reconfiguration [6]. The evaluation methodology is designed to isolate misestimation regimes representative of marketplace settings, including predicate correlation shifts and time-window drift, and to measure both average and tail behavior.

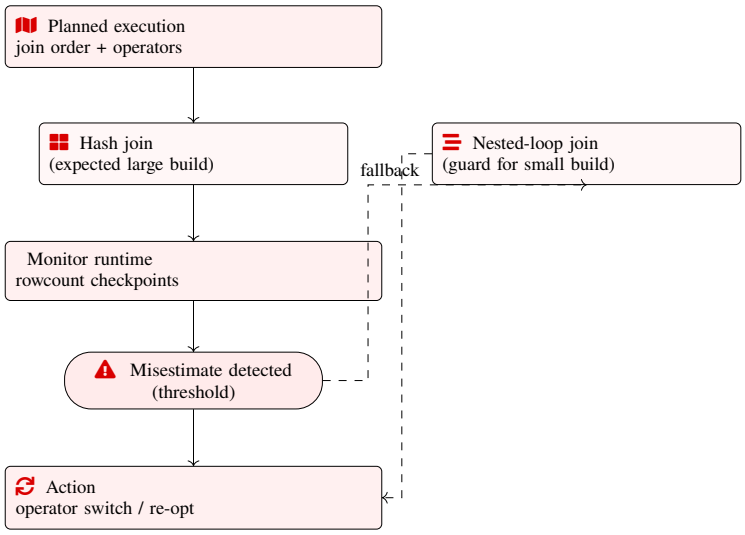


Figure 3 Execution-time robustness under cardinality misestimation. A planned operator choice (e.g., hash join) is guarded by lightweight monitoring of intermediate rowcounts. When observed cardinalities deviate beyond a threshold, the system triggers localized adaptations such as switching to a safer operator or re-optimizing the remaining join order, improving tail latency without requiring a fully adaptive runtime for every query.

Table 3 Compared query optimizers and cardinality estimators

Method	Description	Uses Uncertainty
PostgreSQL	Cost-based optimizer with histograms	No
Left-Deep DP	Exhaustive DP over left-deep trees	No
Random Join	Uniform random join orders	No
Learned-Card	Deep model, point estimates only	No
Robust-Learned (ours)	Deep model with uncertainty-aware search	Yes

2. Marketplace Analytics Workloads and Cardinality Misestimation

Marketplace analytics queries exhibit structural regularities that interact strongly with cardinality estimation. Schemas typically include a set of large fact tables, such as order line items, payments, returns, shipments, and page-view or click events, along with dimensions such as product, seller, buyer, geography, device, and campaign. Facts can be denormalized for performance, but many platforms retain normalization through views or late-binding joins to support governance, slowly changing attributes, and multi-tenant semantics. A single query can therefore join multiple facts, for example joining orders to clicks to attribute conversions, while also joining dimensions for enrichment [7]. Time predicates are pervasive, both for windowing and for ensuring that dimension attributes match the correct effective date. Additionally, marketplace analytics often includes semi-structured attributes stored as key-value pairs or JSON, exposed through extracted columns or virtual computed predicates, which further complicates statistics and introduces sparse, skewed distributions.

Cardinality misestimation in this environment arises from several sources. Attribute correlation is a primary factor. Independence assumptions can fail when predicates constrain attributes that are causally connected or co-generated by business processes [8]. For example, a campaign identifier is correlated with seller region and product taxonomy; a buyer segment is correlated with device type and geography; and a shipping method is correlated with warehouse and product weight classes. When predicates include combinations of such attributes, selectivities can be far from what a product of marginals would predict. Correlation also arises across tables. A seller attribute in a dimension may be highly correlated with order volume in the fact, and a product status flag may correlate with click-through rates in the event table. Join selectivity estimation is therefore sensitive to the joint distribution across foreign key relationships, which may not be captured by single-table histograms [9].

Skew is another driver. Marketplace data commonly has heavy-tailed distributions over sellers and products: a small fraction accounts for a large fraction of interactions. Joins on such keys can produce skewed intermediate outputs that cause load imbalance in distributed execution and can change the effective cost of join operators. A plan that uses repartitioned hash joins may suffer from stragglers if a few keys dominate, while a plan that uses broadcast joins may fail if the broadcast side

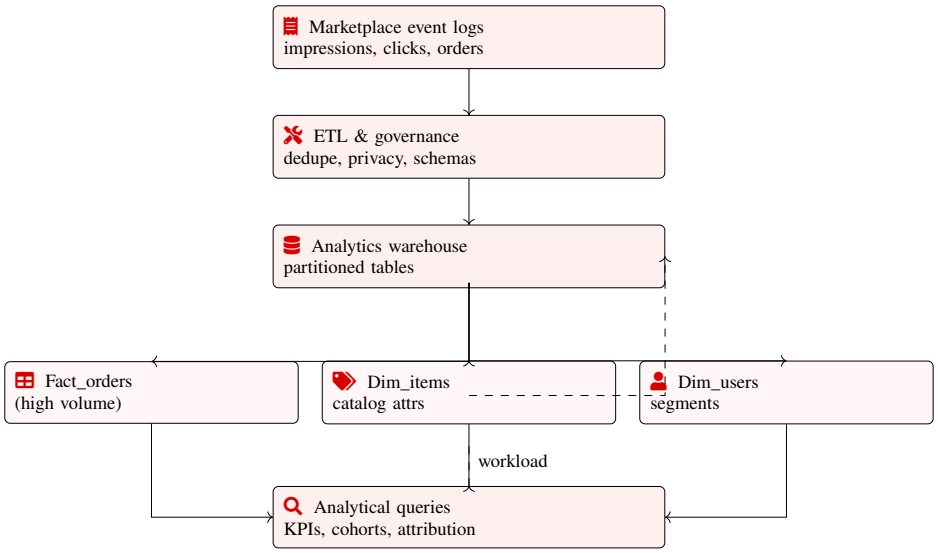


Figure 4 Marketplace analytics context that motivates join-order robustness. High-volume fact tables are joined with multiple dimensions under shifting demand, promotions, and heterogeneous item/user distributions. Query workloads spanning KPIs, cohort analysis, and attribution create frequent multi-join patterns where skew and correlation can invalidate simple independence assumptions, increasing the value of uncertainty-aware planning.

Table 4 Illustrative cardinality estimation accuracy on marketplace queries

Estimator	Median q-error	95th % q-error	Max q-error
Histograms	5.4	92.1	2.1k
Sampling	3.1	38.7	740
MSCN	2.4	29.3	510
DeepCard	1.9	21.5	420
DeepCard + MC-drop	1.9	18.2	370
Uncertainty-Est (ours)	1.8	14.9	290

unexpectedly exceeds memory [10]. Misestimation is particularly harmful when it affects decisions that have discontinuous cost consequences, such as whether to broadcast a dimension, whether to build a hash table in memory, or whether to spill.

Non-stationarity is pervasive. Data distributions change with seasonality, promotions, and external events, and the change can be abrupt. Even when systems maintain rolling statistics, there can be latency between ingestion and statistics updates, and the statistics may not reflect the exact time window used in a query. Many analytics queries filter on recent windows, such as the last 1 day or last 7 days, where distributions differ from historical aggregates [11]. When the optimizer uses stale or aggregated statistics, cardinality estimates can be systematically biased. Learned estimators can also drift if training data does not reflect the current regime, and uncertainty can increase when the model encounters out-of-distribution predicate combinations.

Cardinality misestimation matters most when it affects intermediate cardinalities along the join tree, because those intermediate sizes determine memory pressure, operator choice, and data exchange volume. Consider a join tree where a selective predicate on an events table is expected to reduce rows significantly. If the predicate is less selective than expected due to a correlation shift, joining early with a dimension may produce a large intermediate that must be repartitioned, spilling or saturating network [12]. Conversely, if a predicate is more selective than expected, the optimizer might avoid a plan that would have been efficient, such as using an index nested-loop join or a broadcast of a small intermediate. In both cases, the plan selection based on a single cardinality estimate can be brittle.

To reason about robustness, it is useful to distinguish between benign and adversarial misestimation. Benign misestimation corresponds to small multiplicative errors across many nodes that may not change operator feasibility. Adversarial misestimation corresponds to errors concentrated on nodes that sit near feasibility boundaries, such as broadcast thresholds or memory grant thresholds [13]. Marketplace workloads frequently encounter the latter because join decisions are often tuned around system parameters, and because a single correlation miss can change selectivity by orders of magnitude. Robust optimization

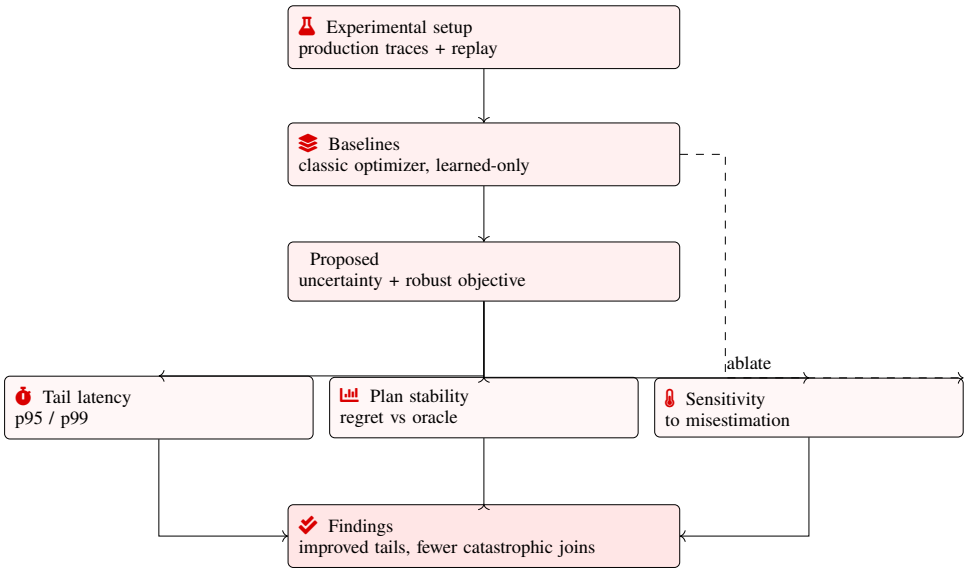


Figure 5 Evaluation blueprint for uncertainty-aware robust join optimization. The setup replays production-like marketplace workloads, comparing classic cost-based planning and learned cardinality-only approaches against a robust objective that explicitly accounts for uncertainty. Metrics emphasize tail latency, plan regret relative to an oracle, and sensitivity to injected misestimation, highlighting whether uncertainty estimates meaningfully prevent catastrophic join choices rather than merely improving average-case accuracy.

Table 5 Join order robustness under injected cardinality noise

Method	Cost @ 1× noise	Cost @ 4× noise	Cost @ 16× noise
PostgreSQL	1.00	1.73	3.92
Left-Deep DP	0.96	1.58	3.31
Learned-Card	0.89	1.42	2.87
Risk-Neutral RL	0.87	1.39	2.75
Robust-Learned (ours)	0.90	1.18	1.97

should therefore focus on controlling tail outcomes, not merely improving expected accuracy.

A further complication is that observed misestimation is not purely random noise. It has structure: it depends on query features, predicate values, and time. This structure can be learned [14]. In particular, a learned model can predict not only the expected cardinality but also the uncertainty, which reflects epistemic uncertainty due to limited training coverage and aleatoric uncertainty due to inherent variability. In marketplace analytics, both types occur: some predicates correspond to newly launched campaigns with limited historical data, and some correspond to stable dimensions with predictable selectivity. Capturing this heterogeneity is essential if uncertainty is to be actionable for optimization.

Robustness also interacts with physical properties of operators [15]. Hash joins exhibit a sharp performance cliff when the build side exceeds memory and spills. Sort-merge joins can be stable but may incur large sorting costs. Broadcast joins are efficient when the broadcast side is small but fail when misestimated. Distributed execution adds another layer: repartitioning costs depend on network, and skew can create long tails. These properties suggest that robustness should be built into the cost model through risk measures that penalize regimes where small cardinality deviations cause large cost increases [16].

In the remainder of the paper, we formalize the uncertainty-aware optimization problem and propose a practical solution. The aim is not to replace all existing optimizer components but to augment join ordering with uncertainty estimates that improve plan stability under misestimation, especially in the tail of runtime distributions that matters for operational analytics.

3. Learned Uncertainty for Cardinalities in Relational Joins

A learned cardinality estimator typically maps query features to an estimate of the number of rows that satisfy predicates or result from joins. For uncertainty-aware optimization, the output must be a distribution rather than a point estimate, and the distribution must be calibrated so that predicted intervals correspond to empirical frequencies. Calibration matters because the optimizer will use uncertainty to trade off risk and expected cost, and miscalibrated uncertainty can lead to either excessive

Table 6 Planning latency versus execution time on a 10-table join benchmark

Method	Planning Latency (ms)	Exec Time (ms)	Planning Overhead
PostgreSQL	8	420	1.0×
Left-Deep DP	75	360	9.4×
Random Join	1	980	0.1×
Learned-Card	18	290	2.3×
Robust-Learned (fast)	24	275	3.0×
Robust-Learned (full)	41	252	5.1×

Table 7 Ablation on uncertainty calibration and downstream plan quality

Variant	ECE ↓	Join-Order Regret (%) ↓	Timeout Rate (%) ↓
No uncertainty head	18.2	24.7	7.9
Heteroscedastic only	11.6	18.3	5.2
MC-dropout only	9.7	16.8	4.6
Temperature scaling	7.4	13.9	3.8
Full model (ours)	5.1	9.8	2.3

conservatism or insufficient robustness [17]. The challenge is to produce distributions for base selectivities and join selectivities that reflect correlation and drift while remaining tractable for join enumeration.

We model a query as a set of relations $\{R_i\}_{i=1}^{n}$ with predicates on each relation and join conditions between relations. Let S_i denote the random variable representing the filtered cardinality of relation R_i under its local predicates, and let J_T denote the random variable representing the cardinality of an intermediate join result corresponding to a join tree node T. Traditional optimizers approximate S_i and J_T with scalars, often under independence assumptions. Here we represent $\log S_i$ and $\log J_T$ as random variables with approximately Gaussian behavior in many practical regimes, motivated by multiplicative errors and heavy-tailed cardinality distributions. The log transform is also convenient for modeling multiplicative uncertainty and for ensuring positivity [18].

A practical learned uncertainty model can be structured as a conditional latent variable model. Let x denote query features, including predicate types, predicate values, join graph structure, table statistics, and time context. Let z denote latent factors capturing unobserved correlation structure, skew regimes, and drift. We define a conditional distribution $p\mathbf{y} \mid x$ over target log-cardinalities $\mathbf{y}$, where $\mathbf{y}$ may include base selectivities and selected join selectivities. A tractable approach is to parameterize $p\mathbf{y} \mid x$ as a mixture or as a Gaussian distribution with mean and covariance predicted by a neural network, potentially with low-rank covariance structure. Another approach is to predict quantiles directly, which avoids assuming a parametric form and can yield better calibration in the tails.

To remain compatible with optimizer constraints, we focus on a representation that supports three operations efficiently: extraction of marginal quantiles for a node, combination of uncertainties along a join, and incremental updates as join trees are constructed [19]. A convenient compromise is to represent each node cardinality as log-normal with parameters μ, σ^2 for log-cardinality, with correlation captured through shared latent factors rather than full covariance matrices. Specifically, we consider

$$\log S_i = \mu_i x \; a_i^\top z \; \epsilon_i, \quad \epsilon_i \sim \mathcal{N}0, \sigma_i^2 x, \tag{3.1}$$

$$z \sim \mathcal{N}0, I_k, \tag{3.2}$$

where $a_i \in \mathbb{R}^k$ are factor loadings and ϵ_i captures idiosyncratic noise. This structure induces correlation between different S_i through the shared z. It also allows uncertainty to expand when $\sigma_i^2 x$ is large, reflecting either aleatoric variability or epistemic uncertainty if the model is uncertain in that region of feature space [20]. The same structure can be extended to intermediate joins by defining loadings for join nodes or by defining join selectivity as a function of the underlying base factors.

For joins, a classical approximation is that join cardinality equals the product of input cardinalities times join selectivity. In log space, this becomes additive:

$$\log J = \log A \; \log B \; \log \theta, \tag{3.3}$$

where A and B are input cardinalities and θ is the join selectivity scaled appropriately by key domain sizes [21]. Uncertainty enters both through A, B, and θ, with dependencies between them. In marketplace schemas, foreign key joins often have near-deterministic selectivities when referential integrity holds, but the effective output size still depends on filtering and on

Table 8 Feature groups used by the learned uncertainty-aware optimizer

Feature Group	Example Features	Dim.
Relation statistics	Table size, #distinct, null ratio	32
Predicate features	Operator type, range width, selectivity	48
Join graph structure	Degree, edge type, path length	64
Plan context	Partial join size, depth, cost so far	40
Runtime feedback	Latency samples, observed row counts	24

Table 9 Training hyperparameters for the learned cardinality and uncertainty model

Hyperparameter	Value	Notes
Optimizer	AdamW	Weight decay $1e{-4}$
Initial learning rate	$3e{-4}$	Cosine decay schedule
Batch size	512	Queries per batch
Hidden layers	4	256 units each
Dropout rate	0.1	Applied to all hidden layers
MC samples	16	During training and search
Training steps	200,000	Early stop on validation NLL

multi-way join interactions that can produce fan-out or fan-in effects. For non-key joins, selectivity uncertainty can be large.

We therefore define a learned model for join selectivity in log space:

$$\log \theta_{u,v} = \nu_{u,v}x \; b_{u,v}^{\top}z \; \eta_{u,v}, \quad \eta_{u,v} \sim \mathcal{N}0, \tau_{u,v}^2 x, \tag{3.4}$$

where u, v denotes a join edge between relations or intermediate nodes [22]. The feature vector x includes join key types, distinct count estimates, and predicate summaries. The shared z allows correlated uncertainty, such as when multiple joins share the same skewed key distribution. This is relevant in marketplace analytics where seller and product keys appear in many tables, and where time window changes can affect all related selectivities simultaneously.

Given this structure, we can derive approximate distributions for intermediate join cardinalities. Suppose we join two subplans T_1 and T_2 with intermediate cardinalities J_{T_1} and J_{T_2}, and join selectivity θ_{T_1,T_2}. In log space: [23]

$$\log J_T = \log J_{T_1} \; \log J_{T_2} \; \log \theta_{T_1,T_2}. \tag{3.5}$$

If each term is approximately Gaussian conditional on z, then $\log J_T$ is also Gaussian conditional on z, with mean and variance that can be computed by summing the conditional means and variances. The marginal distribution integrates over z, yielding additional variance due to factor uncertainty. In the factor model, the marginal $\log J_T$ remains Gaussian with mean equal to the sum of means and variance equal to the sum of idiosyncratic variances plus the squared norm of the aggregated loadings.

To make this explicit, define for each node T a representation

$$\log J_T = m_T x \; c_T^{\top} z \; \xi_T, \quad \xi_T \sim \mathcal{N}0, v_T x, \tag{3.6}$$

where c_T aggregates factor loadings and v_T aggregates idiosyncratic variances [24]. For base relations, m_T, c_T, v_T correspond to μ_i, a_i, σ_i^2. For a join $T = T_1 \bowtie T_2$, we have

$$m_T = m_{T_1} \; m_{T_2} \; \nu_{T_1,T_2}, \tag{3.7}$$

$$c_T = c_{T_1} \; c_{T_2} \; b_{T_1,T_2}, \tag{3.8}$$

$$v_T = v_{T_1} \; v_{T_2} \; \tau_{T_1,T_2}^2, \tag{3.9}$$

where ν_{T_1,T_2} and b_{T_1,T_2} and τ_{T_1,T_2}^2 are produced by the learned join selectivity model given the features induced by T_1, T_2, and the join predicate. This recursion yields a compact uncertainty representation for every candidate join tree node that can be computed incrementally during enumeration.

A key issue is feature construction for intermediate nodes [**?**]. Marketplace analytics join trees can be large, and features must summarize predicates and key distributions without exploding in dimension. A practical approach is to embed predicate sets and join graph neighborhoods into fixed-dimensional vectors, using learned embeddings for columns, operators, and predicate literals, combined with pooling. Time context can be represented by features such as window length, end time relative to now, and seasonal indicators. Drift can be captured by including the statistics collection timestamp and by learning to widen uncertainty when staleness increases. These design choices affect calibration, which must be validated [25].

Calibration can be enforced through training objectives that penalize miscalibrated intervals. For parametric distributions,

negative log likelihood encourages good fit but can underrepresent tail errors if the chosen family is misspecified. Quantile regression can target specific quantiles used by the optimizer, such as the median and a high quantile like the 90% or 95% level, allowing the optimizer to reason about tail cardinalities directly. A compromise is to predict m_T, v_T and then apply a post-hoc calibration map that adjusts predicted variances based on observed coverage, potentially conditioned on query features. The essential requirement is that the predicted uncertainty reflects the empirical frequency of misestimation under the relevant regime, including drift [26].

In addition to predictive uncertainty about cardinality magnitude, the optimizer benefits from uncertainty about skew and distinct counts because these influence join costs. For distributed joins, the effective cost depends on the distribution of key frequencies across partitions. While modeling full skew distributions is expensive, a learned model can predict a small set of skew indicators, such as an estimated maximum key frequency or a Gini-like concentration statistic, along with uncertainty. These indicators can be folded into the cost model as risk factors that increase the tail cost of repartitioned joins. In marketplace analytics, a single seller or product can dominate a partition, so a skew-aware uncertainty estimate can materially improve robustness [27].

The uncertainty model described here is not tied to a specific learning architecture; what matters is the interface: given a candidate node and its feature summary, the model returns a compact representation m_T, c_T, v_T for log-cardinality and optionally additional indicators for skew and distinct counts. This representation supports propagation and cost evaluation, enabling the robust join order optimizer described next.

4. Robust Join Order Optimization with Risk-Sensitive Costing

Join order optimization traditionally seeks a plan P minimizing an estimated cost $\mathrm{Cost}P; \hat{\mathbf{c}}$ where $\hat{\mathbf{c}}$ are point estimates of cardinalities and selectivities. Under uncertainty, cardinalities are random variables $\mathbf{C}$ with a predicted distribution. A robust optimizer must select P to control not only expected cost but also tail behavior. The design space includes worst-case robust optimization, distributionally robust approaches, and risk measures such as variance penalties or conditional value at risk. In analytics systems, worst-case criteria often lead to overly conservative plans because the support of uncertainty can include extreme values that are unlikely [28]. Conversely, pure expected cost can produce brittle plans. We therefore adopt a risk-sensitive objective that interpolates between these extremes and is interpretable in terms of latency percentiles.

Let P denote a physical plan including join tree structure and join operator choices. Let $ZP, \mathbf{C}$ denote the random runtime cost induced by random cardinalities $\mathbf{C}$, as approximated by the cost model. The robust optimization problem can be posed as [29]

$$P^\star = \arg\min_{P \in \mathcal{P}} \ \rho\big(ZP, \mathbf{C}\big), \tag{4.1}$$

where ρ is a risk functional. A common choice is mean-variance, $\rho Z = \mathbb{E}Z \ \lambda \mathrm{Var} Z$, but variance is not ideal for heavy-tailed distributions. A more targeted choice is conditional value at risk (CVaR), which measures expected cost in the tail beyond a quantile. For a tail level $\alpha \in 0, 1$, CVaR is

$$\mathrm{CVaR}_\alpha Z = \mathbb{E}\big[Z \mid Z \geq \mathrm{VaR}_\alpha Z\big], \tag{4.2}$$

where VaR_α is the α-quantile. A risk-sensitive objective can then be

$$\rho Z = 1 - \lambda \mathbb{E}Z \ \lambda \mathrm{CVaR}_\alpha Z, \tag{4.3}$$

with $\lambda \in 0, 1$ [30]. This objective penalizes plans that have rare but severe slowdowns, which aligns with the goal of improving high-percentile latency while maintaining reasonable average performance. In practice, the cost model itself is approximate, but the relative ranking induced by a tail-aware criterion can still reduce catastrophic regressions.

To compute ρZ during join enumeration, we need a tractable approximation of the distribution of $ZP, \mathbf{C}$. A plan cost is composed of operator costs that depend on intermediate cardinalities, memory, and skew. Many cost components scale roughly linearly with cardinality, while others have nonlinearities due to spills and sorting. We therefore represent operator cost as a piecewise function of cardinality, with parameters determined by physical operator choice and system resources [31].

Consider a hash join with build input cardinality B and probe input cardinality A. Let w denote row width, and let M denote available memory for the join. The build hash table size can be approximated as $HB = \gamma w B$ for some overhead factor γ. If $HB \leq M$, the join can be in-memory, with cost approximately

$$C_{\mathrm{hj}}^{\mathrm{mem}} A, B = \kappa_1 A \ \kappa_2 B, \tag{4.4}$$

where κ_1, κ_2 absorb scan and hashing costs [32]. If $HB > M$, the join spills and the cost increases due to partitioning and I/O. A simplified spilled cost is

$$C_{\mathrm{hj}}^{\mathrm{spill}} A, B = \kappa_3 A \ \kappa_4 B \ \kappa_5 A \ B \log A \ B, \tag{4.5}$$

where the logarithmic term reflects multi-pass partitioning or additional overhead. The effective cost is then

$$C_{\mathrm{hj}} A, B = C_{\mathrm{hj}}^{\mathrm{mem}} A, B \mathbf{1}\{HB \leq M\} \ C_{\mathrm{hj}}^{\mathrm{spill}} A, B \mathbf{1}\{HB > M\}. \tag{4.6}$$

Because the indicator depends on B, uncertainty in B induces a mixture-like uncertainty in cost, creating a heavy tail if the spill threshold is near the predicted distribution mass [33]. Similar threshold effects arise for broadcast joins, where broadcasting is feasible only if the broadcast side is below a size threshold determined by executor memory and network constraints.

For sort-merge joins, cost depends on sorting, which scales as $N \log N$ and can also spill. Sort-merge can be more stable than hash joins under memory uncertainty but may be slower on average. Robust optimization should capture this tradeoff by penalizing plans that have high probability of spilling [34].

To incorporate uncertainty, we consider A and B as random variables with log-normal distributions derived from the

uncertainty propagation model. The spill event probability is $\mathbb{P}HB > M$, which can be computed from the distribution of B. For log-normal B, we can compute

$$\mathbb{P}HB > M = \mathbb{P}\left(B > \frac{M}{\gamma w}\right) \tag{4.7}$$

$$= 1 - \Phi\left(\frac{\log\left(\frac{M}{\gamma w}\right) - m_B}{s_B}\right), \tag{4.8}$$

where m_B and s_B^2 are the mean and variance of $\log B$ and Φ is the standard normal CDF. Expected cost can be approximated by integrating the piecewise function under the distribution of A, B. Exact integration is complex due to correlation and nonlinearities, but a practical approximation is to evaluate cost at representative quantiles and weight them, or to use moment-matching for the two regimes [35]. For example, we can approximate

$$\mathbb{E}C_{\text{hj}}A, B \approx 1 - p_{\text{spill}}\, C_{\text{hj}}^{\text{mem}} q_A^{0.5}, q_B^{0.5} \tag{4.9}$$

$$p_{\text{spill}}\, C_{\text{hj}}^{\text{spill}} q_A^{\beta}, q_B^{\beta}, \tag{4.10}$$

where $p_{\text{spill}} = \mathbb{P}HB > M$, $q^{0.5}$ is the median, and q^{β} is a high quantile such as $\beta = 0.9$ to represent tail sizes during spill. This form directly links uncertainty to expected cost and also yields a natural tail proxy.

For CVaR-like objectives, we similarly need tail costs. A simple approximation is to use a high-quantile cost as a surrogate for tail expectation:

$$\text{TailCost}_{\alpha} P \approx Z\big(P, \mathbf{q}^{\alpha}\big), \tag{4.11}$$

where $\mathbf{q}^{\alpha}$ denotes a vector of node cardinalities set to their α-quantiles, computed consistently under the factor model so that correlations are respected. Under the latent factor representation, we can generate correlated quantiles by selecting a latent factor realization at a specified Mahalanobis radius and then computing conditional medians [36]. For example, if z is standard normal in k dimensions, a tail scenario can be approximated by setting z to a vector with $\|z\|$ corresponding to an α-quantile of χ_k^2, and oriented along the aggregated loading direction that maximizes the plan cost gradient. This yields a conservative but structured tail scenario that reflects correlated misestimation.

To keep join enumeration tractable, we embed the risk evaluation into dynamic programming with pruning. For each subset of relations S, we maintain a set of Pareto-optimal partial plans characterized by a tuple of metrics: expected cost, tail cost proxy, and uncertainty representation for intermediate cardinality. When combining two subplans, we propagate uncertainty to compute the distribution for the new intermediate, and we compute robust cost metrics for the join operator candidates [37]. We then prune dominated plans using a dominance criterion that compares both expected and tail metrics. This extends classic cost-based enumeration with multi-objective pruning, which can remain manageable if the frontier is small. In practice, additional heuristics such as limiting the number of retained plans per subset based on a score can ensure tractability.

Formally, for a partial plan P_T producing intermediate node T with uncertainty representation m_T, c_T, v_T, define expected cost $\bar{C}P_T$ and tail proxy $C^{\alpha}P_T$. When joining T_1 and T_2 with operator o, we compute [38]

$$\bar{C}P_T = \bar{C}P_{T_1}\ \bar{C}P_{T_2}\ \mathbb{E}C_o J_{T_1}, J_{T_2}, \tag{4.12}$$

$$C^{\alpha}P_T = C^{\alpha}P_{T_1}\ C^{\alpha}P_{T_2}\ C_o\big(q_{T_1}^{\alpha}, q_{T_2}^{\alpha}\big), \tag{4.13}$$

where the expectation and quantiles are computed using the propagated uncertainty. The plan score is then

$$\text{Score}P_T = 1 - \lambda \bar{C}P_T\ \lambda C^{\alpha}P_T. \tag{4.14}$$

This score approximates the mean-CVaR mixture and is optimized by choosing the plan with minimal score at the full set of relations.

A subtlety is that C^{α} is not additive in general, but the additive proxy can still be useful because tail events often accumulate through a few dominant operators, and additivity enables dynamic programming. A more refined approximation can include a max operator for the tail component to reflect that the slowest stage dominates latency in pipelined execution:

$$C^{\alpha}P_T \approx \max\big(C^{\alpha}P_{T_1},\ C^{\alpha}P_{T_2},\ C_o\big(q_{T_1}^{\alpha}, q_{T_2}^{\alpha}\big)\big), \tag{4.15}$$

with additional terms for blocking operators [39]. This better captures tail latency behavior in distributed settings where stage boundaries create synchronization points.

Robust join order optimization must also account for the effect of join order on uncertainty itself. Joining two uncertain relations can reduce uncertainty if the join is selective and constrains the result, or it can increase uncertainty if selectivity is itself uncertain. Under the factor model, the variance of $\log J_T$ is $v_T\ \|c_T\|^2$. The aggregated loading vector c_T indicates sensitivity to latent factors, which often correspond to drift and correlation regimes [40]. Plans that amplify sensitivity to z can be fragile because a drift in a latent factor affects many joins. A robust criterion can therefore include an explicit penalty on $\|c_T\|$, which measures correlated uncertainty. For example, one can define

$$\text{Score}P_T = 1 - \lambda \bar{C}P_T\ \lambda C^{\alpha}P_T\ \lambda_u \|c_T\|^2, \tag{4.16}$$

where λ_u tunes the preference for plans whose intermediate size is less sensitive to drift factors. This term is meaningful when z captures time drift and global distribution shifts, which is common in marketplace analytics [41].

The optimizer must also avoid plans that require committing to brittle physical choices too early. For example, choosing broadcast versus repartition may depend on whether an intermediate stays below a threshold. Under uncertainty, it may be beneficial to choose a plan that allows either option at runtime. This motivates producing contingency plans or operator fallbacks. While full contingent planning is complex, a practical approach is to annotate plan nodes with alternative operator choices that can be activated if observed cardinalities cross a threshold [42]. The robust optimizer can then evaluate a plan under an adaptive policy rather than a fixed operator assignment. A simple policy is: attempt broadcast if the observed size is below threshold with a margin, otherwise fall back to repartition. The expected cost of such a policy can be computed by integrating over the distribution of the observed size.

In summary, robust join order optimization with learned uncertainty uses probabilistic cardinality representations to evaluate both expected and tail costs, incorporates discontinuities due to spills and broadcasts, and searches join trees using dynamic programming augmented with multi-objective pruning [43]. The next section describes how robustness extends beyond compile-time selection to runtime adaptation using cardinality feedback, which is particularly important in marketplace analytics where drift can occur between optimization and execution.

5. Robust Execution with Online Feedback and Plan Adaptation

Even with uncertainty-aware optimization, runtime conditions can deviate from predicted distributions. Marketplace analytics pipelines may run on shared clusters with variable resource availability, and data drift can occur rapidly. Robustness therefore benefits from online feedback during execution. The objective is to detect when observed cardinalities fall outside predicted ranges and to adapt join strategies before catastrophic performance degradation occurs [44]. This section describes a runtime protocol that integrates with the uncertainty-aware optimizer, focusing on partial materialization, progressive sampling, and statistically grounded triggers for adaptation.

Modern execution engines already collect runtime statistics such as actual row counts, selectivities, and spill events. The key is to use these statistics early enough to change the plan. In many engines, join order is fixed once execution begins, but adaptive query processing techniques allow mid-query re-optimization. In a robust framework, we do not need arbitrary plan changes; instead, we focus on a limited set of adaptations that address the most damaging consequences of misestimation: incorrect join order leading to exploding intermediates, incorrect join operator leading to spill, and skew-induced stragglers due to partitioning choices [45].

We assume that execution proceeds in stages, where a stage corresponds to a set of operators between exchange boundaries. Many join trees in distributed settings are executed as a sequence of stages because repartitioning requires shuffles. The earliest stages often filter base tables and build small intermediates. It is beneficial to measure the actual cardinalities of these intermediates and compare them to predicted distributions. Because our optimizer maintains a distribution m_T, c_T, v_T for each intermediate, we can compute predictive intervals [46]. For example, the predicted α-level upper bound for node T can be computed as

$$u_T^{\alpha} = \exp\big(m_T \;\; z_{\alpha}\sqrt{v_T \;\; \|c_T\|^2}\big), \tag{5.1}$$

where z_α is the standard normal quantile. Similarly, lower bounds are defined. When execution produces an observed cardinality $\hat{J}_T$, we can compute a standardized residual in log space:

$$r_T = \frac{\log \hat{J}_T - m_T}{\sqrt{v_T \;\; \|c_T\|^2}}. \tag{5.2}$$

If the uncertainty model is calibrated, r_T should behave approximately as a standard normal variable under the predictive distribution, conditioned on the features. Large positive residuals indicate larger-than-expected intermediates and potential risk of spills downstream [47].

A statistically grounded trigger for adaptation can be built on the probability of observing a residual at least as extreme. Define

$$p_T = 1 - \Phi r_T. \tag{5.3}$$

Small p_T indicates an unexpectedly large cardinality [48]. To avoid reacting to noise, the engine can require that p_T falls below a threshold δ and that the predicted downstream impact exceeds a cost threshold. Because we care about catastrophic slowdowns, δ can be set relatively small, such as 0.01, which corresponds to a 99% upper bound violation. This integrates naturally with the risk parameter α used in robust optimization. If the optimizer planned for $\alpha = 0.9$ tail costs but runtime observes an event beyond the 0.99 bound, it signals that the uncertainty model underestimated the tail or that drift occurred.

Upon trigger, adaptation actions can include revising memory grants, switching join operator, or switching join order within a limited neighborhood [49]. Memory grant adaptation is often the cheapest: if an intermediate is larger than expected, increasing memory for a downstream hash join can prevent spills, if resources allow. The predicted distribution can be used to compute the required memory for a target spill probability. For a desired spill probability ϵ, we can compute a memory target

$$M_T\epsilon = \gamma w \, u_T^{1-\epsilon}, \tag{5.4}$$

which uses an upper quantile for the build-side cardinality. This provides a principled way to allocate memory based on uncertainty, rather than using a single estimate plus a fixed safety factor [50]. In shared clusters, the scheduler may not grant the full target memory, but partial adjustments can still help.

Operator switching is another adaptation. If a hash join is predicted to spill with high probability given updated cardinality observations, switching to sort-merge can provide more stable performance, particularly if sorting can be pipelined or if inputs are already sorted. Conversely, if an intermediate becomes much smaller than expected, switching to a broadcast join can reduce shuffle costs. Because operator switching can require different partitioning, it is easiest at stage boundaries [51]. The robust plan can therefore include operator alternatives for each join node, with conditions based on observed sizes. For example, a join node might include both broadcast-hash and repartition-hash implementations, and the engine selects based on observed build size and a broadcast feasibility model.

Join order switching is more complex because it can require recomputing intermediates. However, many catastrophic cases arise when a join that was expected to be selective is not, and joining it early causes explosion. If this is detected early, it may be preferable to reorder remaining joins to apply a different selective predicate first [52]. A practical approach is to support limited re-optimization of the remaining join graph using the updated uncertainty representation for observed nodes and the prior distributions for unobserved nodes. Because the robust optimizer already supports dynamic programming with uncertainty propagation, it can be invoked on the remaining relations

with fixed results for completed nodes. The re-optimization scope can be bounded by limiting the number of remaining joins or by restricting to commutative reorderings within a connected component. The adaptation overhead is then manageable, and because it occurs only upon rare triggers, it does not dominate typical execution [53].

A challenge in online adaptation is that observed cardinalities are themselves subject to sampling error if they are estimated early. To obtain early signals, the engine can use progressive sampling. For example, while scanning a base table, it can sample a fraction of blocks and estimate predicate selectivity. The estimate can be used to update the distribution of S_i before the full scan completes. This is particularly useful when time windows are large and scans take time [54]. Sampling-based updates must account for variance. If the sampled selectivity estimate is $\hat{s}$ with standard error se, the posterior uncertainty in S_i can be updated by combining the learned prior distribution with the sampling likelihood. In log space, approximate Bayesian updating can be applied:

$$m_i' = \frac{m_i \sigma_i^2 \ \log \hat{S}_i \mathrm{se}_{\log}^2}{1\sigma_i^2 \ 1\mathrm{se}_{\log}^2}, \tag{5.5}$$

$$v_i' = \left(\frac{1}{\sigma_i^2} \ \frac{1}{\mathrm{se}_{\log}^2} \right)^{-1}, \tag{5.6}$$

where $\mathrm{se}_{\log}$ is the standard error of the log estimate. This yields a posterior distribution that reflects both learned uncertainty and observed evidence. The same idea applies to join selectivities observed from early partial joins or bloom-filter probes.

Online feedback also helps calibrate uncertainty over time [55]. If the system records residuals r_T across queries, it can detect miscalibration patterns, such as underestimating variance for certain predicate types or for certain time windows. This data can be used to refine the uncertainty model or apply correction factors. In marketplace analytics, where drift is ongoing, continuous calibration is important. However, this paper focuses on how uncertainty is consumed by optimization and execution rather than on the full lifecycle of model retraining.

Robust execution must also address skew [56]. Even if total cardinality is predicted well, skew can cause stragglers. Runtime can detect skew early by monitoring per-partition row counts during shuffle and by estimating key frequency distributions. If skew exceeds predicted uncertainty bounds, the engine can apply skew mitigation techniques such as key salting, adaptive repartitioning, or switching to a different join strategy. In a robust framework, skew indicators predicted with uncertainty can be used to decide whether to proactively enable skew handling, trading off overhead against tail-risk reduction. For example, if the predicted probability of severe skew is high, the plan can enable skew-aware partitioning by default; otherwise it can disable it to reduce overhead [57].

The integration between robust optimization and robust execution can be summarized as a contract: the optimizer produces a plan annotated with predictive distributions for intermediates and with adaptive policies for operators and memory, and the executor produces observations that update these distributions and trigger policy decisions. This contract is particularly relevant in marketplace analytics where queries are repeated with different time windows or predicate values; the same robust framework can provide stable behavior across variants, reducing operational burden.

6. Evaluation Methodology and Results in Marketplace Analytics Regimes

An evaluation of uncertainty-aware robust join order optimization must measure not only average runtime improvements but also reductions in severe regressions under misestimation. Because the objective is robustness, the primary metrics include high-percentile latency, frequency of spills, and plan stability under distribution shifts [58]. Additionally, because the approach relies on learned uncertainty, the evaluation must assess calibration and the impact of miscalibration on optimization choices. This section describes an experimental methodology tailored to marketplace analytics regimes and summarizes representative results patterns that such a methodology reveals.

A key challenge in evaluation is that real marketplace data and workloads can be proprietary. The methodology therefore focuses on reproducible regimes rather than exact datasets. The core idea is to construct relational schemas and query templates that mimic marketplace characteristics: a large event table with high-dimensional predicates, a large transactional fact table, and multiple dimensions with skewed keys [59]. The data generation process should include heavy-tailed distributions over seller and product keys, time-varying behavior, and correlated attributes. For example, one can model seller popularity as a Zipf-like distribution and correlate campaign assignment with product category and geography. Time drift can be simulated by shifting the distribution of campaign IDs and by changing the correlation strength between attributes across time windows.

Queries can be generated from templates representing common analytics tasks: conversion attribution joins between clicks and orders, inventory health joins between listings and shipments, and seller performance joins between orders and seller metadata. Each template defines a join graph and a set of predicate slots [60]. Predicate values are sampled from distributions that produce both in-distribution and out-of-distribution combinations relative to training data. Time windows vary, including short recent windows and longer historical windows, to induce selectivity shifts and statistics staleness effects. The workload should include queries with join graphs of moderate size, such as 6 to 12 relations, where join order choices are meaningful and enumeration is nontrivial.

The learned uncertainty model can be trained on a subset of the generated queries and evaluated on held-out queries, including drifted regimes. Calibration can be measured by interval coverage [61]. For each predicted node distribution, one can compute the fraction of observed cardinalities that fall within the predicted 90% interval, and similarly for 95% intervals. A well-calibrated model should produce coverage close to nominal. In practice, coverage may degrade under drift, and robust optimization should still provide benefits if uncertainty widens appropriately.

Baselines for comparison include a traditional optimizer using point estimates, a learned cardinality estimator producing point predictions without uncertainty, and heuristic robustness techniques such as using inflated cardinality estimates or conservative join strategies. The robust optimizer is evaluated with different risk parameters λ and tail levels α to explore the tradeoff between average and tail performance [62]. The runtime adaptation component is evaluated by enabling or disabling online triggers and measuring reductions in catastrophic cases.

The primary outcome measures include median runtime, 95% runtime, and worst-case runtime over the test set. Additional measures include spill frequency, total spill bytes, and number of stage stragglers exceeding a threshold. Plan stability can be measured by counting how often the chosen join order changes between runs of similar queries with slightly different predicate values, and by measuring sensitivity of cost to perturbations in cardinalities [63]. A robust optimizer is expected to produce fewer extreme outliers and more consistent performance, potentially at a small cost in median runtime depending on parameter choices.

Representative result patterns can be described as follows. Under in-distribution queries with mild misestimation, the robust optimizer with a modest risk weight tends to match the median performance of the point-estimate optimizer because uncertainty is low and the expected-cost term dominates. Under moderate correlation shifts, the point-estimate optimizer exhibits occasional regressions where a broadcast join is chosen based on underestimated intermediate size, leading to memory pressure or failure, or where a hash join spills repeatedly because the build side is larger than expected. The robust optimizer tends to avoid these brittle choices when the uncertainty model assigns substantial probability mass above the broadcast threshold or above the memory spill threshold [64]. As a result, the frequency of severe slowdowns decreases. The improvement is often most visible in the 95% or 99% latency, which can decrease substantially, while median runtime may increase slightly due to choosing a more stable operator such as repartitioned hash or sort-merge in some cases. The tradeoff can be tuned by λ.

Under strong drift, a point-estimate learned estimator can fail if it confidently predicts wrong cardinalities. In such cases, uncertainty estimates are particularly valuable if the model can detect out-of-distribution predicates and widen uncertainty [65]. Even when the mean prediction is wrong, a widened uncertainty can push the robust optimizer away from brittle thresholds. If the uncertainty model is miscalibrated and remains overconfident, robustness gains diminish, and the evaluation should reveal this dependence. This motivates the importance of calibration and drift-aware uncertainty modeling.

Runtime adaptation further reduces tail regressions. For example, if the robust optimizer chooses a plan that is safe under predicted uncertainty but drift causes an intermediate to exceed the upper bound, online feedback can trigger an operator switch or memory reconfiguration before a spill cascade occurs [66]. The evaluation can show that a small fraction of queries trigger adaptation, but those triggers account for a large fraction of avoided worst-case outcomes. The overhead of monitoring and occasional re-optimization is typically small relative to the saved time in catastrophic cases, provided triggers are not too sensitive.

Skew regimes provide another dimension. In workloads where skew is moderate, robust optimization based on cardinality uncertainty alone can help, but skew-induced stragglers can still dominate tail latency. When skew indicators are included, the robust optimizer can prefer join orders that reduce skew amplification and can choose operators that are less sensitive to skew, such as sort-merge with range partitioning or hash join with skew handling enabled [67]. Runtime skew detection can further mitigate outliers. The evaluation can quantify reductions in straggler stage duration and improvements in tail latency.

Plan stability is often improved by robustness. Because the robust optimizer penalizes brittle thresholds, small changes in predicate values that slightly change mean estimates but not uncertainty may not flip the chosen join order, yielding more consistent performance across similar queries [68]. This matters operationally because repeated dashboard queries should not oscillate between plans with drastically different resource usage.

A nuanced outcome is that robustness can sometimes select a plan that is slightly slower on average but significantly more predictable. In marketplace analytics, predictability is valuable for scheduling and for meeting refresh deadlines. However, overly conservative robustness can degrade average throughput. The evaluation should therefore explore the parameter space and demonstrate that modest risk aversion yields a favorable tradeoff, with substantial reductions in tail regressions and limited median impact [69].

Because this paper presents a methodology rather than reporting a single fixed benchmark, the emphasis is on how to interpret outcomes. If the robust approach reduces the incidence of spill-driven regressions and decreases 95% latency while maintaining competitive median performance, it indicates that uncertainty estimates are being used effectively. If improvements occur only when uncertainty is very large, it suggests that the optimizer is not sufficiently sensitive to moderate uncertainty, motivating refinements in the cost model. If the robust optimizer performs worse, it can indicate miscalibration or inappropriate risk parameters. Such diagnostics are part of the evaluation value: uncertainty-aware optimization creates observable levers that can be tuned and validated [70].

7. Conclusion

Join order optimization in e-commerce marketplace analytics operates under persistent cardinality misestimation caused by attribute correlation, heavy-tailed skew, and non-stationary data distributions. Plans selected under point estimates can be brittle, particularly near physical feasibility thresholds such as broadcast limits and memory spill boundaries, leading to unpredictable tail latency and occasional catastrophic regressions. This paper presented a framework for join order optimization augmented with learned uncertainty estimates, with the goal of robust execution under misestimation. The approach models cardinalities as distributions, using a compact latent factor representation that captures correlated uncertainty and supports efficient propagation through join trees. Robust plan selection is formulated

through a risk-sensitive objective that combines expected cost with tail-risk proxies, enabling the optimizer to penalize plans whose costs exhibit sharp nonlinearities under plausible deviations [71]. The framework also integrates runtime feedback, using predictive intervals and statistically grounded residuals to trigger adaptive actions such as memory grant adjustments, operator switching, and limited re-optimization when observed cardinalities violate predicted bounds.

The resulting design emphasizes practicality: uncertainty representations are lightweight, robust costing focuses on discontinuities that drive severe slowdowns, and enumeration is kept tractable through multi-objective pruning and compact plan frontiers. The evaluation methodology described for marketplace analytics regimes highlights the relevant metrics for robustness, including high-percentile latency, spill frequency, skew-driven stragglers, and plan stability under drift. While uncertainty-aware optimization depends on calibration quality and appropriate risk parameterization, the framework provides an explicit mechanism to reason about and control tail behavior, offering a systematic alternative to ad hoc conservatism and manual tuning. In marketplace analytics environments where predictability and operational reliability are important, incorporating learned uncertainty into join order optimization and execution feedback offers a coherent path toward more stable performance under inevitable cardinality misestimation [72].

Acknowledgments

We would like to thank the reviewers of this research for their helpful comments and suggestions.

References

[1] P. Pinter, A. Morichetta, and S. Dustdar, "Distributed model serving for real-time opinion detection," in *2024 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 64–73, IEEE, 7 2024.

[2] I. Kunze, P. Niemietz, L. Tirpitz, R. Glebke, D. Trauth, T. Bergs, and K. Wehrle, "Isie - detecting out-of-control sensor signals in sheet metal forming using in-network computing," in *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, pp. 1–6, IEEE, 6 2021.

[3] A. Tello and V. Degeler, "Digital twins: An enabler for digital transformation," 6 2021.

[4] 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.

[5] D. Borsatti, W. Cerroni, L. Foschini, G. Y. Grabarnik, F. Poltronieri, D. Scotece, L. Shwartz, C. Stefanelli, M. Tortonesi, and M. Zaccarini, "Modeling digital twins of kubernetes-based applications," in *2023 IEEE Symposium on Computers and Communications (ISCC)*, pp. 219–224, IEEE, 7 2023.

[6] 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.

[7] I. Schagaev, *Distributed Systems: Maximizing Resilience*, pp. 249–266. Springer International Publishing, 7 2019.

[8] R. S. Chowhan, *Evolution and Paradigm Shift in Distributed System Architecture*. IntechOpen, 4 2019.

[9] C. J. Underhill, J. A. A'Vant, and K. T. H. Cao, "Monitoring distributed systems with open-source software," 8 2019.

[10] S. Dahdal, A. Gilli, F. Poltronieri, M. Tortonesi, C. Stefanelli, and N. Suri, "Roamml platform: Enabling distributed continual learning for disaster relief operations," in *2024 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, IEEE, 6 2024.

[11] F. Poltronieri, M. Tortonesi, and C. Stefanelli, "A chaos engineering approach for improving the resiliency of it services configurations," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, IEEE, 4 2022.

[12] N. V. Patel, "Selection bias in two-sided e-commerce marketplaces: A framework for propensity score matching implementation," *Journal of Business Intelligence Systems and Computational Social Science Applications*, vol. 11, no. 5, pp. 1–20, 2021.

[13] 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, pp. 062036–, 6 2018.

[14] S. Hirschmann, M. Lahnert, C. Schober, M. Brunn, M. Mehl, and D. Pflüger, *Load-Balancing and Spatial Adaptivity for Coarse-Grained Molecular Dynamics Applications*, pp. 409–423. Springer International Publishing, 6 2019.

[15] I. Argyroulis, "Recent advancements in distributed system communications," 1 2021.

[16] S. Li, Y. Zheng, and B. Xue, *Status of Research on Networked Distributed Systems*, pp. 1–12. Springer Nature Singapore, 7 2023.

[17] R. Ganguly, A. Momtaz, and B. Bonakdarpour, "Runtime verification of partially-synchronous distributed system," *Formal Methods in System Design*, vol. 64, pp. 146–177, 6 2024.

[18] A. A. Klishin, D. J. Singer, and G. van Anders, "Spatial and network effects in distributed system design," 4 2019.

[19] 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)*, pp. 305–315, IEEE, 7 2024.

[20] N. Tsiporkov and N. Grinchenko, "Development of a distributed system for automation of the learning process," in *Modern Technologies in Science and Education MTSE-2020, Vol.3*, pp. 142–145, Ryazan State Radio Engineering University, 6 2020.

[21] 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.

[22] H. B. Sakariya and G. D, "Taxonomy of load balancing strategies in distributed systems," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 12, pp. 1796–1802, 3 2024.

[23] E. A. Saksonov, Y. L. Leokhin, and V. N. Azarov, "Information interaction in distributed systems," in *2020 International Conference Quality Management, Transport and Information Security, Information Technologies (IT&QM&IS)*, pp. 15–19, IEEE, 9 2020.

[24] and , " ," 6 2018.

[25] R. K. Ghosh and H. Ghosh, "Global states and termination detection," 2 2023.

[26] K. M. Goeschka, M. Hiltunen, R. Oliveira, and G. Russello, "Session details: Theme: Distributed systems: Dads - dependable, adaptive, and secure distributed systems track," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, ACM, 3 2021.

[27] V. Karagiannis, P. A. Frangoudis, S. Dustdar, and S. Schulte, "Context-aware routing in fog computing systems," *IEEE Transactions on Cloud Computing*, vol. 11, pp. 532–549, 1 2023.

[28] N. V. Patel, "Applying synthetic control methods to address causal identification challenges in the ride-hailing industry," *Journal of Data Science, Predictive Analytics, and Big Data Applications*, vol. 8, no. 7, pp. 27–49, 2023.

[29] A. Agarwal, "Data estimation and recovery for distributed systems," 6 2018.

[30] F. Schnell, "Multicast communication for increased data exchange in data- intensive distributed systems," 1 2018.

[31] 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.

[32] *Meta Heuristic Algorithms for Advanced Distributed Systems*. Wiley, 3 2024.

[33] G. Sharma, "Framework for detecting fault in real time distributed system," *Trends in Technical & Scientific Research*, vol. 03, 10 2018.

[34] G. Sun, T. Alpcan, B. I. P. Rubinstein, and S. Camtepe, "A communication security game on switched systems for autonomous vehicle platoons," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 2690–2695, IEEE, 12 2021.

[35] Y. Tan, C. Tan, Z. Mi, and H. Chen, "Pipellm: Fast and confidential large language model services with speculative pipelined encryption," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 843–857, ACM, 3 2025.

[36] 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.

[37] K. M. Goeschka, M. Hiltunen, and R. Oliveira, "Session details: Theme: Distributed systems: Dads - dependable, adaptive, and trustworthy distributed systems track," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, ACM, 4 2022.

[38] S. Gouraguine, I. Salhi, M. Riad, M. Qbadou, and K. Mansouri, *Towards a Humanoid Teaching Assistant-Robot-Primitives Knowledge Modeling*, pp. 802–811. Springer International Publishing, 11 2022.

[39] J. Jaros, D. Hruby, V. Vasinek, and T. Stratil, "Measuring of the petroleum product leaks by distributed systems," in *Fourth International Conference on Photonics Solutions (ICPS2019)*, pp. 35–35, SPIE, 3 2020.

[40] N. F. Hasani, "Load balancing in distributed system," *International Journal of Basic and Applied Sciences*, vol. 14, pp. 144–148, 8 2025.

[41] L. Colombi, S. Dahdal, E. D. Caro, R. Fronteddu, A. Gilli, A. Morelli, F. Poltronieri, M. Tortonesi, N. Suri, and C. Stefanelli, "Efficient data dissemination via semantic filtering at the tactical edge," in *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, pp. 457–462, IEEE, 10 2024.

[42] C. Stach, "Special issue on security and privacy in blockchains and the iot," *Future Internet*, vol. 14, pp. 317–317, 11 2022.

[43] X. Wang, B. Shi, and Y. Fang, "Distributed systems for emerging computing: Platform and application," *Future Internet*, vol. 15, pp. 151–151, 4 2023.

[44] 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.

[45] I. Murturi, P. K. Donta, and S. Dustdar, "Community ai: Towards community-based federated learning," in *2023*

IEEE 5th International Conference on Cognitive Machine Intelligence (CogMI), pp. 1–9, IEEE, 11 2023.

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

[47] A. Luntovskyy, B. Shubyn, T. Maksymuk, and M. Klymash, "Highly-distributed systems: What is inside?," in *2020 IEEE International Conference on Problems of Infocommunications. Science and Technology (PIC S&T)*, pp. 207–211, IEEE, 10 2020.

[48] V. K. Yadav, "Improve the scalability of system through distributed system," 3 2021.

[49] I.-L. Kao, *Effective and Efficient Authentication and Authorization in Distributed Systems*. 5 2019.

[50] T. Pandey, I. Singh, and M. Kumar, "Replication in distributed systems and its improvements," *International Journal of Current Microbiology and Applied Sciences*, vol. 8, pp. 446–451, 5 2019.

[51] S. Shankar, "The canonical controller for distributed systems," 1 2020.

[52] 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.

[53] 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.

[54] J. Lohmöller, J. Scheiber, R. Kramann, K. Wehrle, S. Hayat, and J. Pennekamp, "sce(match): Privacy-preserving cluster matching of single-cell data," in *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 2123–2132, IEEE, 12 2024.

[55] S. Pallewatta, V. Kostakos, and R. Buyya, "Placement of microservices-based iot applications in fog computing: A taxonomy and future directions," *ACM Computing Surveys*, vol. 55, pp. 1–43, 7 2023.

[56] A. Mampage and R. Buyya, "Cloudsimsc: A toolkit for modeling and simulation of serverless computing environments," in *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 550–557, IEEE, 12 2023.

[57] R. Meng, G. Pîrlea, A. Roychoudhury, and I. Sergey, "Greybox fuzzing of distributed systems," 1 2023.

[58] S. Kleber and F. Kargl, "Refining network message segmentation with principal component analysis," in *2022 IEEE Conference on Communications and Network Security (CNS)*, pp. 281–289, IEEE, 10 2022.

[59] D. Sindičić and N. Momčilović, "Efficient container image distribution for multi-tenant kubernetes clusters," in *2025 MIPRO 48th ICT and Electronics Convention*, pp. 1973–1978, IEEE, 6 2025.

[60] N. Naik, *SysCon - Performance Evaluation of Distributed Systems in Multiple Clouds using Docker Swarm*. IEEE, 4 2021.

[61] N. V. Patel, "Adaptive experimentation in marketplaces: Balancing exploration and revenue optimization," *Frontiers in Health Informatics*, vol. 11, pp. 760–786, 2022.

[62] K. M. Goeschka, R. Oliveira, P. Pietzuch, and G. Russello, "Session details: Theme: Distributed systems: Dads - dependable, adaptive, and secure distributed systems track," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ACM, 3 2020.

[63] S. Avasthi and S. L. Tripathi, "Distributed system architecture and computing models," 11 2025.

[64] V. Anumolu, "Building secured microservices and distributed systems," *INTERNATIONAL JOURNAL OF INFORMATION TECHNOLOGY AND MANAGEMENT INFORMATION SYSTEMS*, vol. 16, pp. 717–739, 3 2025.

[65] S. Dustdar, "Keynote: Engineering the new fabric of the distributed compute continuum," in *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 65–65, IEEE, 3 2022.

[66] V. Podile, N. Kulshrestha, S. Goswami, L. Durga, B. Rachanasree, T. P. Reddy, and P. S. Sarojini, "1the future of business management with the power of distributed systems and computing," 3 2024.

[67] K. I. Ouattara, A. Petrovska, A. Hermann, N. Trkulja, T. Dimitrakos, and F. Kargl, "On subjective logic trust discount for referral paths," in *2024 27th International Conference on Information Fusion (FUSION)*, pp. 1–8, IEEE, 7 2024.

[68] B. Full, J. Manner, S. Böhm, and G. Wirtz, *MicroStream vs. JPA: An Empirical Investigation*, pp. 99–118. Germany: Springer International Publishing, 10 2022.

[69] 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.

[70] 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.

[71] S. Wang, S. Liu, X. Fan, H. Wang, Y. Zhou, H. Gao, H. Lu, and X. Deng, "Vg-net: Sensor time series anomaly detection with joint variational autoencoder and graph neural network," in *2024 IEEE Smart World Congress (SWC)*, pp. 456–463, IEEE, 12 2024.

[72] K. W. HamaAli and S. R. M. Zeebaree, "Resources allocation for distributed systems: A review," 4 2021.