uOttawa

Robust Query Optimization in Relational Databases based on Approximate Probabilistic Machine Learning

by

Seyed Mohammad Amin Kamali

A thesis submitted to the University of Ottawa

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Digital Transformation and Innovation

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Examining Committee

The following served on the Examining Committee for this thesis.

External Examiner: **Khuzaima Daudjee**

Professor, Cheriton School of Computer Science,

University of Waterloo

Internal Member(s): **Bijan Raahemi**

Professor, Telfer School of Management,

University of Ottawa

Iluju Kiringa

Professor, School of Electrical Engineering and

Computer Science, University of Ottawa

Daniel Amyot

Professor, School of Electrical Engineering and

Computer Science, University of Ottawa

Supervisor: **Verena Kantere**

Professor, School of Electrical Engineering and

Computer Science, University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those concerning consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

Query processing systems rely on a query optimizer to explore alternative execution plans for a given query and choose the one expected to be optimal. The optimizer estimates costs using parameters such as cardinalities that are unknown at compile time, necessitating reliance on estimated values subject to errors. Traditional optimizers target expected optimality without considering the inherent uncertainties in these estimates, frequently resulting in suboptimal runtime performance when estimation errors occur. This thesis addresses this fundamental limitation by formalizing the concept of robustness in query optimization and developing a novel approach called Roq (Robust Query Optimization) based on approximate probabilistic machine learning.

Within the context of Digital Transformation and Innovation, database management systems serve as critical technological infrastructure that underpins modern data-driven initiatives. As organizations increasingly rely on analytics for decision support, the reliability and efficiency of database systems become paramount for successful digital modernization efforts. The robustness of query execution directly impacts the stability of applications and services that fuel digital transformation across sectors.

We first establish a theoretical framework that decomposes cost model uncertainty into distinct components: plan risk (uncertainty inherent to the plan structure) and estimation

risk (uncertainty in cost estimates due to modeling limitations). Building on this foundation, we propose a principled methodology for quantifying suboptimality risk, defined as the likelihood of a plan being suboptimal at runtime compared to alternatives. This risk quantification framework employs variational inference techniques to capture both aleatoric (data) and epistemic (model) uncertainties.

The contributions of this work include: (1) a novel learned cost model architecture that leverages Graph Neural Networks to capture complex structural characteristics of queries and plans, (2) risk-aware plan evaluation strategies that utilize uncertainty quantification to select robust plans, and (3) RobOpt, a practical workload optimization tool that demonstrates the real-world applicability of our theoretical framework, with a human-centered approach.

Our extensive experimental evaluation across multiple benchmarks (JOB, CEB, DSB, and TPC-DS+) demonstrates that Roq consistently outperforms state-of-the-art approaches. The risk-aware plan selection strategies improve the 99th percentile of suboptimality by up to 44.4% compared to existing learned optimizers while improving the average runtime performance by up to 23.0%. Moreover, Roq exhibits exceptional robustness to workload shifts, maintaining stable performance where baseline methods degrade significantly. Analysis of computational overhead confirms that the benefits of risk-aware optimization can be realized with minimal compilation-time impact, making the approach practical for

real-world deployment.

By addressing uncertainty explicitly rather than ignoring it, this research establishes a foundation for more predictable, reliable, and efficient database performance, thus supporting the broader objectives of digital transformation by enhancing the technological backbone upon which modern data-centric innovations depend. The integration of advanced machine learning techniques with traditional database systems exemplifies how cross-disciplinary approaches can address longstanding challenges in the digitalization landscape.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Verena Kantere, for her invaluable guidance, encouragement, and support throughout my doctoral studies. Her insightful advice, patience, and unwavering belief in my abilities have shaped both this thesis and my growth as a researcher.

I am also sincerely thankful to my committee members, Professor Bijan Raahemi and Professor Iluju Kiringa, for their valuable time, thoughtful feedback, and constructive discussions that helped strengthen this work throughout the research process.

I extend special appreciation to Professor Daniel Amyot, who not only taught me the fundamentals of conducting a proper literature review in his Seminar class but also provided in-depth, detailed comments on this thesis that led to significant improvements. His dual contributions as both educator and examiner have been invaluable to my academic development.

I wish to extend my appreciation to my collaborators, Baoming Chang, Chang Liu, and Ning Wang, for their contributions, stimulating conversations, and continuous support during this journey.

A special thanks to my mentors, Calisto Zuzarte and Vincent Corvinelli, whose wisdom, guidance, and encouragement have had a lasting impact on my professional development.

I am equally grateful to the entire IBM Db2 AI Optimizer team for their collaboration, insights, and the enriching experience of working alongside them.

This thesis would not have been possible without the support, generosity, and inspiration of each of these individuals and groups. I am truly grateful for their contributions to my academic and personal growth.

Dedication

This thesis is dedicated to my parents, whose endless love, sacrifices, and values have shaped the person I am today.

To my wife, Zahra, whose patience, kindness, and unwavering support have been my anchor through every challenge of this journey.

This work belongs as much to them as it does to me.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Full Form
ACID	Atomicity, Consistency, Isolation, Durability
AGM	Atserias-Grohe-Marx (bound)
AJA	Adaptive Join Algorithm
ALECE	Attention-based Learned Cardinality Estimator
AM	Adjacency Matrix
ASO	Average Suboptimality
AutoCE	Automatic Cardinality Estimation
BAO	Bandit optimizer for Adaptive Optimization
BN	Bayesian Network
BNN	Bayesian Neural Network
CDF	Cumulative Distribution Function

List of Abbreviations (continued)

Abbreviation	**Full Form**
CEB	Cardinality Estimation Benchmark
CGS	Column Group Statistics
CI	Cardinality-Integral
CORDS	Correlation Detection System
CS	Cardinality-Slope
DB	Database
DBMS	Database Management System
DeepDB	Deep Database
DP	Dynamic Programming
DRL	Deep Reinforcement Learning
DSB	Decision Support Benchmark
ELBO	Evidence Lower Bound
ESS	Error Selectivity Space
ETV	Execution Time Variance
FACE	Flow-based Accurate Cardinality Estimator
FAUCE	Fast and Accurate deep Uncertainty for Cardinality Estimation

List of Abbreviations (continued)

Abbreviation	**Full Form**
FCNN	Fully Connected Neural Network
FSPM	Factorized Sum-Product Network
GJA	Generic Join Algorithm
GMM	Gaussian Mixture Model
GNN	Graph Neural Network
IAM	Integrated Autoregressive Model
IMDB	Internet Movie Database
JGMP	Join Graph Message Passing
JOB	Join Order Benchmark
KDE	Kernel Density Estimation
KL	Kullback-Leibler (divergence)
LEO	Learning Optimizer
LEON	Learning-based query optimizer
LERO	Learning to Rank query Optimizer
LOGER	Learned Optimizer with Guaranteed Error bounds on Regressions
LPCE	Learning-based Progressive Cardinality Estimation

List of Abbreviations (continued)

Abbreviation	**Full Form**
MAE	Masked Autoencoder
MC	Monte Carlo
MCTS	Monte Carlo Tree Search
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSO	Maximum Suboptimality
MSCN	Multi-Set Convolutional Network
Neo	Neural Optimizer
NeuroCard	Neural Cardinality estimator
NNGP	Neural Network Gaussian Process
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PACE	Poisoning Attacks on Cardinality Estimation
PARQO	Penalty-Aware Robust Query Optimization
PCF	Parametric Cost Function
PDF	Probability Density Function

List of Abbreviations (continued)

Abbreviation	**Full Form**
POP	Progressive Query Optimization
RDBMS	Relational Database Management System
REQO	Robust and Explainable Query Optimization
RI	Referential Integrity
RID	Row Identifier
RIO	Robust Optimizer
RLCM	Risk-aware Learned Cost Model
RNN	Recurrent Neural Network
ROQ	Robust Query Optimization
RSPN	Relational Sum-Product Network
RTOS	Reinforcement Tree-based Optimizer for Select
SERF	Selectivity Error Resistance Factor
SIT	Statistics on Intermediate Tables
SNGP	Spectral Normalized Neural Gaussian Process
SOR	Suboptimality Risk
SPJ	Select-Project-Join

List of Abbreviations (continued)

Abbreviation	**Full Form**
SPN	Sum-Product Network
SQL	Structured Query Language
SVD	Singular Value Decomposition
TCNN	Tree Convolutional Neural Network
TPC	Transaction Processing Performance Council
TPC-DS	Transaction Processing Performance Council-Decision Support
UAE	Unified AutoEncoder
VI	Variational Inference
VLDB	Very Large Data Base

List of Notations

Symbol	Description
$\theta, \theta^*, \hat{\theta}, \hat{\theta}_t$	Model parameters (true, estimated, sampled at iteration t)
$x, x^*, \hat{x}$	Input variable or cardinality (true, estimated)
$y, y^*, \hat{y}$	Target variable or execution time (true, predicted)
μ, μ_x	Mean value (general, of variable x)
$\hat{\mu}, \hat{\mu}_t$	Predicted mean (general, at iteration t)
σ, σ_x	Standard deviation (general, of variable x)
σ^2, σ_x^2	Variance (general, of variable x)
$\hat{\sigma}, \hat{\sigma}^2$	Predicted standard deviation and variance
$\hat{\sigma}_t^2, \hat{\sigma}_d^2, \hat{\sigma}_m^2$	Predicted variance (at iteration t, data uncertainty, model uncertainty)
$f_\theta(x), f_{\theta^*}(x)$	Cost function with parameters θ and true parameters θ^*

List of Notations (continued)

Symbol	Description
$p(y^* \mid x^*, D), p(\theta \mid D)$	Predictive and posterior distributions
$p(y^* \mid x^*, \theta)$	Likelihood function
$q(\theta)$	Variational distribution of model parameters θ
$q(y^* \mid x^*)$	Approximate predictive distribution of y^* given x^*
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$\chi^2(d.o.f = n)$	Chi-squared distribution with n degrees of freedom
$E[\cdot]$	Expectation operator
$E_{\theta \sim q(\theta)}[\cdot]$	Expectation operator with respect to distribution $q(\theta)$
$\mathrm{Var}(\cdot)$	Variance operator
$\mathrm{KL}(\cdot \parallel \cdot)$	Kullback-Leibler divergence
ELBO	Evidence Lower Bound
$\int \cdot d\theta$	Integration with respect to parameter θ
$\sum_{t=1}^{T}$	Summation from $t = 1$ to T
$\mathbf{AM}_{n \times n}$	Adjacency matrix of size $n \times n$
$\mathbf{I}_{2 \times m}, \mathbf{A}_{m \times d}$	Edge index matrix and edge attribute matrix
$\mathbf{v}_{ij}, \mathbf{u}_i$	Feature vectors (edge between nodes i, j, node i)

List of Notations (continued)

Symbol	Description
$\mathbf{0}$	Zero vector
$D, \mathcal{D}$	Training dataset and data samples
$\mathcal{X}, \mathcal{X}^*$	Input space or cardinality space (estimated, true)
T, N, n, m, d	Counts (Monte Carlo samples, data points, tables, edges, feature dimensions)
$\|R\|, \|S\|$	Cardinality of relations R and S
T_i	Table i or execution time at iteration i
P, p_i	Query execution plan (general, plan i)
$c_P(q), c_{opt}(q)$	Cost of executing plan P and optimal plan at cardinality point q
$ET(p_i)$	Execution time of plan p_i
$\text{SubOpt}(P, q)$	Suboptimality of plan P at cardinality point q
$\text{SubOpt}(p_i)$	Suboptimality of plan p_i at the true cardinality point
$\text{MSO}(P), \text{ASO}(P)$	Maximum and Average Suboptimality of plan P across the entire cardinality space
$\text{SERF}(q_e, q_a)$	Selectivity Error Resistance Factor

List of Notations (continued)

Symbol	**Description**
λ	Cost adjustment parameter (hashing overhead, index lookup cost, or user-defined threshold)
$\bowtie, \bowtie^{NL}, \bowtie^{HS}$	Join operators (general, nested-loop, hash)
$\text{Card}(T_i)$	Cardinality of table T_i
Q_{error}, Loss	Q-error metric and loss function
$\alpha, \beta, \gamma, \delta$	Learning parameters or coefficients
π	Mathematical constant

Chapter 1

Introduction

This introductory chapter establishes the foundation for robust query optimization in database management systems within the context of digital transformation. Section 1.1 examines the strategic importance of database systems across various stakeholder perspectives: database developers who face high engineering costs (Section 1.1.1), database administrators who shoulder complex maintenance tasks (Section 1.1.2), and end-users whose experience directly impacts business outcomes (Section 1.1.3). Section 1.2 provides an overview of query optimization, explaining its components (Section 1.2.1), execution plans (Section 1.2.2), and evaluation methods (Section 1.2.3) before formally defining the concept of robustness in this domain (Section 1.2.4). Through illustrative motivating examples (Section 1.2.5.1), the chapter demonstrates how cost inaccuracies can lead to

suboptimal performance and establishes the problem that this research addresses (Section 1.3). The chapter concludes by detailing the contributions to the field (Section 1.4), listing related publications (Section 1.5), and outlining the structure of the thesis (Section 1.6). The research methodology based on Design Science Research paradigm is presented in Chapter 2. Together, these sections frame the research problem while providing the necessary context for understanding the significance of the proposed solutions.

1.1 Digital Transformation and Robust Query Optimization and Processing

Database management systems are arguably the technological backbone of the Digital Economy. Any piece of software used in digital modernization efforts is integrated and relies on a back-end database to store, update, retrieve, and analyze data. This reliance is more profound in areas where data fuel transformation efforts. This includes initiatives that rely on descriptive, predictive, and prescriptive analytics as transformation drivers. Descriptive analytics refers to data-centered initiatives and technologies that enable the organization to explain past events and uncover patterns. Predictive analytics goes one step further to make inferences and predictions for future events. Prescriptive analytics involves using descriptive and predictive analytics to produce and suggest the best course

of action that is expected to lead to desirable outcomes in the future. All three types of analytics heavily depend upon reliable databases to operate as expected.

The reliability of database systems goes well beyond satisfying the ACID[1] properties that are intended to guarantee data validity despite errors, power failures, and other mishaps. It also involves providing efficiency and robustness as two main characteristics of query execution performance. While the former refers to the average runtime performance, the latter is characterized by limiting the system's worst-case performance. Over the past several years, an extensive body of research has been devoted to addressing these two challenges. The classical approaches are not typically adopted in practice due to various limitations. In recent years, new methods based on Artificial Intelligence (AI), as a Digital Transformation enabler, have emerged that aim at solving these problems while mitigating the shortcomings of the classical methods. AI-based approaches are being suggested to solve problems such as plan enumeration, plan costing, cardinality estimation, index selection, workload management, query rewrite, and more. While most of these studies aim at improving the average performance, a more recent stream of research has focused on improving the robustness of the system.

Against these backdrops, DBMSs play a dual role in the realm of Digital Transformation. They serve as indispensable enablers by providing the foundation for data-driven

[1] Atomicity, Consistency, Isolation, Durability

initiatives and analytics that drive innovation. Simultaneously, DBMS itself undergoes a transformative process, adapting to the evolving technological landscape through the integration of AI.

This thesis focuses on the role of DBMS as a subject of Digital Transformation with the goal to make it more efficient and robust in the face of demanding use cases. More specifically, query optimization as the main culprit for determining the query execution performance is the focus of the current thesis. By taking a human-centered approach, the following sections take a closer look at the problems faced by various DBMS stakeholders that motivate and define the objectives of any transformation initiatives in query optimization. This will follow by an introduction to query optimization and its robustness, motivating examples, problem definition, and a summary of contributions in the rest of this chapter.

1.1.1 High Costs of Developing and Maintaining Query Optimizers – DB Developer Perspective

Building a good query optimizer takes thousands of person-engineering-hours and is "an art only a few experts fully master" [101]. For example, the cost model is carefully handcrafted for each type of access, join, or aggregate operator by formulating a special logic that

explains its cost. These formulations rely on various parameters that are either estimated or assumed to take certain values. This makes the cost model design susceptible to limitations that may influence its effectiveness in certain situations where the estimated parameters may be inaccurate or the assumptions may not hold. Understanding why the optimizer picks a bad plan, therefore, requires one to be fully aware of such limitations, either by having direct knowledge of the design or extensive experience in diagnosing bad plans. In addition, the design needs to be tediously maintained as the system engine specifications evolve. As a result, none of the freely available open-source query optimizers come close to the commercial ones offered by tech giants [101]. Yet, even commercial optimizers usually suffer from bad search space and inaccurate cardinality and cost estimation, thus leading to poor query performance [156].

By the introduction of modern AI-based components over the past years, their adoption in practice has been slow. One reason is due to the risk they pose to the other components in the database that are not designed to interact with the newly invented module. Adding new modules, therefore, requires expensive tests to ensure a net positive impact on the system's performance and stability. In addition, while AI-based techniques promise a reduction in development efforts, their maintenance and tuning may become yet another source of complexity for an already sophisticated system.

Against these backdrops, a modernization of the query optimizer necessitates the fol-

lowing from the perspective of the database developers:

1. Replace the heuristics-based components that require human-led reconfiguration as new system architecture and hardware emerge, with counterparts that adapt to the changes automatically

2. Have minimal configuration and tuning overheads such that the benefit of their adoption for the development teams outweighs their costs

3. Preferably target modules with minimal cascading effects on the other modules of the query optimizer to minimize the risks of instability

1.1.2 Database Maintenance Overheads – DB Admin Perspective

The meticulous complexity of the DBMS, and the query optimizer in particular, mandates constant tuning and maintenance by highly skilled database administrators. These are especially demanding tasks in face of reading and writing large volumes of data by several end-points and running ever-changing analytical queries against the database. The inter-dependencies of various modules and their configurations makes these tasks even more challenging. Performance tuning is yet one of many duties carried out by a DB Admin,

such as setting up databases, loading and maintaining the data, manage the hardware infrastructure to ensure scalability, managing the risks of failure to various mishaps, securing the database and managing credentials and groups, and many more. In addition, even the most skilled admins sometimes struggle to find optimal configuration that provide consistent performance for all workloads and needs. Adopting a configuration that improves the performance for majority of cases but causes only a few catastrophic regressions is usually not acceptable.

Furthermore, organizations typically do not have the resources to hire as many skilled DB admins as needed to meet their increasing needs for managing their data infrastructure. This can easily overload the few available resources and lead to either burnout of staff or under-performance of the data infrastructure.

From a database admin perspective, this backdrop necessitates:

1. Automating database maintenance tasks as much as possible,
2. Make the databases adapt to the changing data, workload, and computing environment automatically, and
3. Adopt methods and configurations that achieve an acceptable average performance while minimizing risks of regression.

1.1.3 Robustness as a Prerequisite of a Successful Modernization Process – The End-User and Business Perspectives

Many database modernization efforts that aim at improving the performance, only do so by average. They usually have no means to avoid or at least limit the regressions. This translates to inconsistent experience for the end-user. While most queries see benefits, there can be some that take much longer to complete than they should. To demonstrate this impact, let us consider an e-commerce scenario, in which the website loading times are of utmost importance to the end-user. Studies have shown remarkable relationships between website load times and conversion rates [2]. One study demonstrated in a case study that the conversion rate exceeded 20% for loading times up to 0.3 second, but decreased to as little as 10% with just a 0.5 second increase in loading times [136]. According to a recent study [153], it was found that in business-to-business (B2B) situations, a website that has a loading time of less than 1 second experiences conversion rates that are three times higher compared to a website that takes 3 seconds to load. Similarly, in business-to-consumer (B2C) scenarios, reducing the loading time from 5 seconds to 1 second results in a 2.5 times increase in the conversion rate.

While website loading time can be influenced by multiple factors, these figures empha-

[2]The ratio of the users who "convert" to customers or show the desired behavior after visiting a website.

size the critical significance of database response times. The speed at which the database responds directly affects the overall system latency, which in turn affects the user experience and ultimately impacts the bottom lines for businesses.

From an end-user and a business perspective, these considerations necessitate:

1. Improving the average performance, while minimizing the risk of regressions to the extent possible, and

2. Not adversely impacting the systems overall stability

Guided by this human-centered approach, the remainder of this thesis demonstrates how robust query optimization can address the specific requirements of these stakeholder groups. In Chapter 6, we validate our proposed methods through extensive experiments on diverse benchmarks including JOB, CEB, DSB, and TPC-DS+, which represent a range of real-world and synthetic workloads. These experiments measure both prediction accuracy and runtime performance, with particular emphasis on robustness to workload shifts—a critical concern for all stakeholders. Further, Chapter 7 introduces RobOpt, a practical workload optimization tool that implements our theoretical framework. RobOpt's architecture is explicitly designed with modules that address the unique needs of each persona, as detailed in Section 7.4, demonstrating how uncertainty-aware cost modeling can be ef-

fectively integrated into database systems to benefit database developers, administrators, and end-users alike.

1.2 Query Optimization

1.2.1 Overview

The concept of a Relational Database was first introduced by Edgar F. Codd [26], who was a researcher at the IBM San Jose Research Laboratory. The design aimed to shield users from the complexities of data. Later, Chamberlin and Boyce [15] introduced the Structured Query Language (SQL) as a domain-specific language to retrieve and manipulate data in Relational Database Management Systems (RDBMSs). SQL allows users to specify 'what' to do, leaving the system to determine 'how' to do it.

In query processing, high-level SQL expressions are translated into low-level procedural expressions for optimization and execution. Query optimization, a major component of query processing, consists of two phases:

1. **Logical Optimization**: This phase involves transforming the original low-level query expression (often written in relational algebra) into alternative expressions

using equivalency rules and heuristics, often cost-based. This is also known as query rewrite.

2. **Physical Optimization**: This phase constructs and selects the query execution plan.

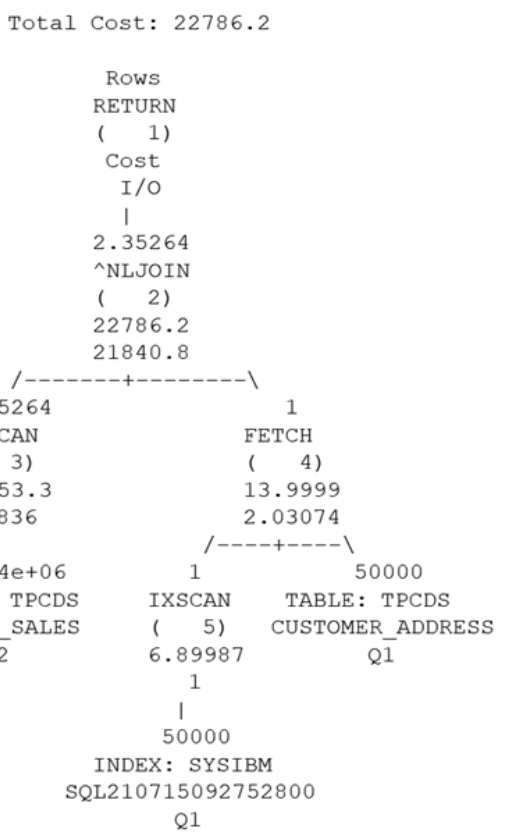
1.2.2 Query Execution Plan

A query execution plan specifies the order of operations, the physical implementation of join and access operators, and the location of execution, particularly in partitioned databases. The optimizer generates alternative plans, estimates their execution costs, and selects the least expensive plan for execution. This process, known as plan optimization, involves constructing a search space of potential plans using techniques like dynamic programming, exhaustive search, or greedy approaches.

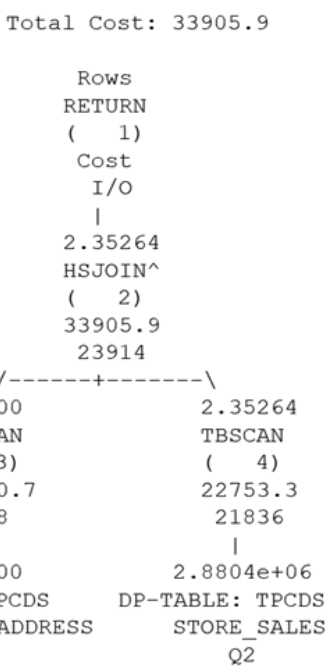
Query execution plans can be represented as tree graphs, where each node corresponds to an operator (e.g., table access, filter, join, aggregation). All query plans produce equivalent outputs but differ in execution costs. A coarse cost model, using data statistics and system environment specifications, evaluates alternative plans. For example, Figure 1.1 illustrates a query from the TPC-DS dataset with two alternative plans and their associated costs in Db2.

```
SELECT
  TPCDS.CUSTOMER_ADDRESS.CA_ADDRESS_SK2,
  TPCDS.STORE_SALES.SS_PROMO_SK
FROM
  TPCDS.CUSTOMER_ADDRESS
  INNER JOIN TPCDS.STORE_SALES
  ON TPCDS.STORE_SALES.SS_ADDR_SK =
TPCDS.CUSTOMER_ADDRESS.CA_ADDRESS_SK
WHERE
  +00110.90 = TPCDS.STORE_SALES.SS_COUPON_AMT AND
  +00206.50 <= TPCDS.STORE_SALES.SS_NET_PAID;
```

(a)



(b)



(c)

Figure 1.1: An example query based on TPC-DS dataset and two alternative plans

Building an effective query optimizer requires significant expertise. Hand-crafted cost models estimate the cost of various operators, but inaccuracies in parameter estimation or assumptions can lead to suboptimal plans. Commercial optimizers often outperform open-source ones but still face challenges such as poor search space exploration and inaccurate cost estimation [101, 158].

1.2.3 Plan Evaluation

The goal of the query optimizer is to identify the plan with the minimum expected execution time. This is achieved using search algorithms, such as Dynamic Programming, which traverse the search space and estimate the costs of partial plans at each step. Cost models guide this process by estimating the processing requirements of operators in the plans.

For example, a simple cost model, C_{out}, accumulates the estimated cardinality of the output stream of each operator [130]:

$$C_{\text{out}}(T) = \begin{cases} |R| & \text{if } T = R \\ |T| + C_{\text{out}}(T_1) + C_{\text{out}}(T_2) & \text{if } T = T_1 \bowtie T_2 \end{cases} \tag{1.1}$$

In this formula, T represents a table or intermediate result, R denotes a base table, $|R|$ indicates the cardinality (number of rows) of table R, and $|T|$ represents the cardinality

of table T. The join operation $T_1 \bowtie T_2$ represents a join between tables T_1 and T_2. The cost function $C_{\text{out}}(T)$ recursively computes the total cardinality by summing the output cardinality of the current operation with the costs of its sub-operations, providing a simple yet effective way to estimate the computational overhead of query execution plans.

This basic model, while limited, can be extended. For instance, C_{mm} differentiates between hash joins and index-nested-loops joins, assuming all processing is in memory [88]:

$$C_{\text{mm}}(T) = \begin{cases} \tau \cdot |R| & \text{if } T = R \vee T = \sigma(R) \\ |T| + |T_1| + C_{\text{mm}}(T_1) + C_{\text{mm}}(T_2) & \text{if } T = T_1 \bowtie_H T_2 \\ C_{\text{mm}}(T_1) + \lambda \cdot |T_1| \cdot \max\left\{\frac{|T_2 \bowtie_R T_1|}{|T_1|}, 1\right\} & \text{if } T = T_1 \bowtie_{NL} T_2 \end{cases} \tag{1.2}$$

Here, R is a base table, $\tau \leq 1$ discounts the cost of base table scans, and $\lambda \geq 1$ adjusts for the cost of index lookups. While real-world databases employ more sophisticated models that consider I/O and memory usage, parameters like τ and λ must be carefully tuned to align estimated costs with actual execution times.

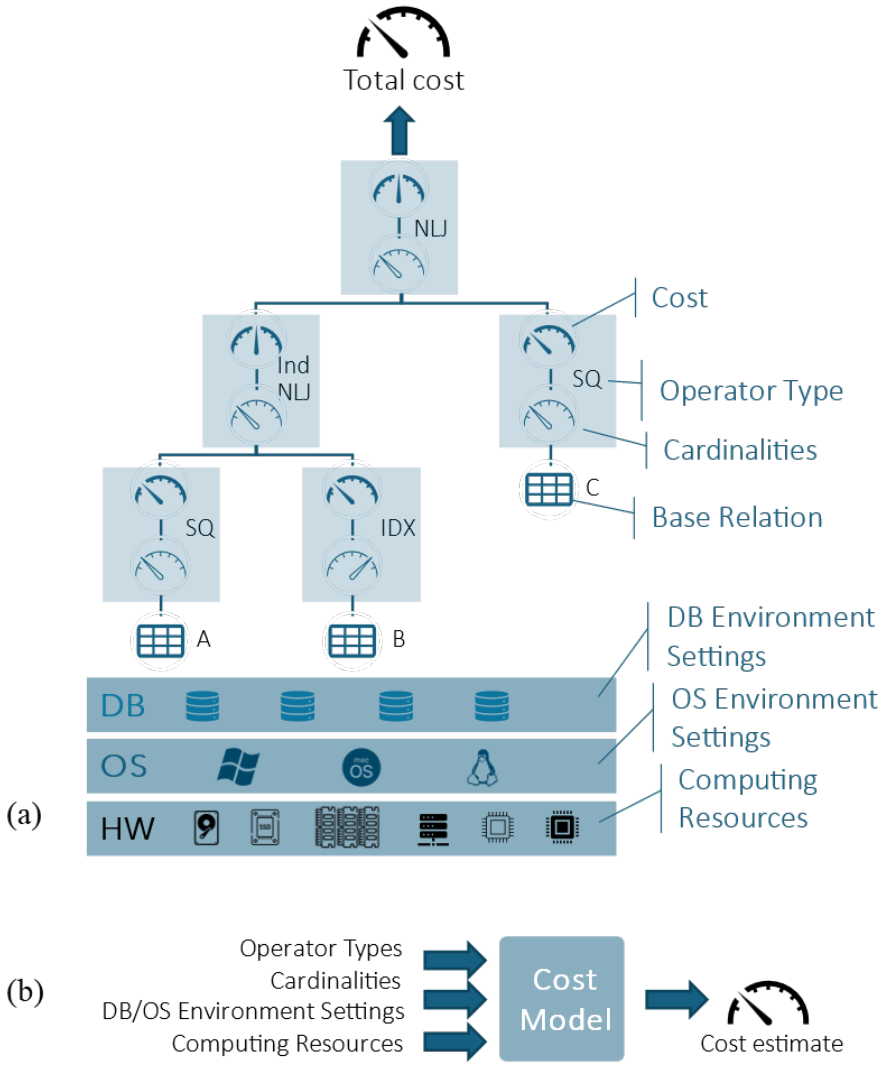


Figure 1.2: Plan evaluation (a) parameters impacting plan's cost (b) a cost model that takes these parameters as input and estimates the cost

1.2.4 Robustness in Query Optimization

State-of-the-art query optimizers primarily rely on *expected optimality*—selecting the query plan that minimizes the expected cost. In this process, the plan's optimality is determined based on estimated costs, assuming that these estimates are sufficiently accurate. However, this simplifying assumption can lead to significant problems, particularly when the estimated costs are inaccurate. The optimizer may then choose a query plan that is *non-robust* (i.e., *risky*), which can result in suboptimal performance in real-world executions. This issue is further illustrated by the motivating example discussed in Section 1.2.5.1.

1.2.4.1 What is Robustness in Query Optimization?

Robustness in query optimization refers to a system's ability to perform well despite uncertainty and variability in the execution environment. While this concept has been studied extensively in recent years [165], a universally accepted formal definition remains elusive. A recent study by Haritsa [52] proposed a framework for defining robustness in query optimization systems, identifying four key characteristics:

1. **Bounded Maximum Suboptimality:** A robust system ensures that its performance does not degrade beyond a certain threshold, even in the worst-case scenario.

In this sense, robustness can be enhanced by improving worst-case performance, sometimes at the expense of average-case performance.

2. **Graceful Performance Degradation:** Robust systems exhibit gradual performance loss as environmental factors (e.g., database size, workload complexity) change. In contrast, non-robust systems may experience *performance cliffs*, where small changes in the environment lead to drastic performance drops. Therefore, a suboptimal plan that degrades gracefully may outperform an optimal one that suffers steep performance losses.

3. **Scalability:** A robust query optimization approach should handle increasing workload complexity, database size, and distributional skew without sacrificing performance. This characteristic ensures that the system can handle both simple and complex queries effectively.

4. **Theoretical Guarantees:** Robust methods provide strong theoretical guarantees about their performance relative to the ideal. This involves ensuring that the method can provide bounds on performance, such as maximum suboptimality.

Haritsa [52] also emphasizes that robustness is application-dependent, and the specific characteristics of robustness may vary based on the context. In relational database management systems (RDBMS), robustness can be evaluated at various levels of granularity,

including individual plan operators, the overall query plan, or the entire query execution process.

1.2.4.2 Existing Robust Query Processing Approaches

Several approaches have been proposed to address the problem of robustness in query optimization, but they come with significant drawbacks:

- **Uncertainty Quantification via Intervals:** Some studies have attempted to quantify uncertainties in parameter estimations by using intervals rather than point estimates [7, 104, 110]. While these methods provide some flexibility, they still require re-optimization and introduce significant compilation overheads, as they necessitate the modification of the execution plan with additional operators.

- **Run-time Discovery Mechanisms:** Other studies have attempted to reduce the dependency on parameter estimation altogether by employing run-time discovery mechanisms [32, 76, 77]. While these mechanisms avoid relying on inaccurate estimates, they still come with high overhead and may result in suboptimal plans, especially in systems with limited resources.

- **Robustness Quantification Using PCF Derivatives:** One approach for quantifying robustness [154] involves analyzing the derivative and the area under the

cost function curve with respect to estimated cardinalities. However, this method assumes that the cost function is deterministic, which overlooks the probabilistic nature of execution time. It also relies on analytical cost models that make simplifying assumptions that may not hold in practice [23].

1.2.4.3 Challenges with Current Approaches to Robust Query Processing

Achieving robustness in query optimization is particularly challenging due to the sensitivity of cost estimates to inaccurate parameter estimations. For instance, errors in estimating selectivity—such as overestimating or underestimating the number of matching records in a predicate—can lead to inaccurate cost estimations for the corresponding operators. These errors are propagated and sometimes amplified through the query plan, potentially resulting in large discrepancies between the estimated and actual costs.

This issue is further complicated by the fact that non-robust plans may experience substantial cost overruns when execution occurs with different parameter values than those estimated at compile-time. In contrast, a robust plan remains relatively stable across a broader range of parameter values. However, traditional query optimization methods, which rely on precise parameter estimations, are vulnerable to the inherent uncertainty of real-world data.

Despite years of research into cardinality estimation and parameter prediction, accu-

rately estimating these parameters remains an unsolved problem. Even state-of-the-art cardinality estimation methods can produce highly inaccurate results in certain cases.

1.2.4.4 Recent Advances and Limitations in Learning-based Methods

In recent years, learning-based methods have gained significant attention for their ability to estimate query execution times directly at runtime, thereby bypassing the need for traditional, error-prone parameter estimation [57, 101, 103, 135]. These approaches leverage machine learning models to predict execution costs, showing notable promise in improving robustness, particularly when dealing with errors in input parameters. For instance, Neo [101] demonstrates robustness to cardinality estimation inaccuracies. This robustness arises from the capacity of ML models to adjust their reliance on noisy or uninformative input features, dynamically tuning their weights to prioritize more reliable signals.

Despite these advancements, learning-based methods face several limitations that hinder their effectiveness in robust query optimization. One major challenge lies in managing *uncontrolled predictive uncertainties*. While ML models can generalize well, their predictions often carry inherent variability due to factors such as model overfitting, data sparsity, or distributional shifts between training and runtime environments. These uncertainties can lead to suboptimal query plans being selected, as the models do not provide explicit guarantees about their predictions.

Another critical limitation is the absence of explicit mechanisms for quantifying the *robustness* of query plans or the predictive models themselves. Robustness entails assessing the risk associated with choosing one query plan over its alternatives, which requires understanding the full probabilistic distribution of estimated costs. Current learning-based approaches primarily focus on optimizing for expected values, neglecting the variability or uncertainty inherent in these estimates. This gap makes it challenging to evaluate the risk of suboptimal performance and undermines the ability to make informed plan selections.

Addressing these limitations necessitates a more principled approach to robustness in learning-based methods. Specifically, incorporating techniques to model and quantify predictive uncertainties, as well as frameworks to assess the likelihood of suboptimality across query plans, is crucial for advancing this field.

1.2.4.5 A New Approach to Robustness: Quantifying Risk and Uncertainty

Inspired by recent advancements in machine learning for managing uncertainties [40], this thesis proposes a novel and principled definition of robustness in query optimization. This approach builds upon recognizing the probabilistic nature of execution times and estimated costs. By quantifying the risks associated with query plan selection, we aim to provide a more reliable method for selecting robust plans in uncertain environments.

1.2.5 Motivating Example

Cost estimation in query optimization is inherently uncertain, regardless of the method employed. These uncertainties are not uniform across different query plans or subplans, as the propagation of estimation errors through plan operators varies significantly. Such variations result in different levels of uncertainty at the roots of alternative plans [154], as illustrated by examples in Section 1.2.5.1. This issue becomes particularly problematic when two alternative plans have very similar estimated costs, making the optimizer's choice more susceptible to inaccuracies.

1.2.5.1 Visualizing Cost Inaccuracies

Figure 1.3 illustrates various scenarios for plan cost inaccuracies. Each subplot depicts the probability density function (PDF) of the estimated costs for two alternative plans: the *expected optimal plan* and the *alternative plan*. For simplicity, the costs are assumed to follow a normal distribution. While real-world costs are often skewed, transformations (e.g., log transformations) can approximate them to normal distributions, facilitating analysis.

These distributions represent the range of possible values the estimated costs may take across the parameter space. The optimizer typically selects the plan with the minimum expected cost as the *optimal plan*. However, the alternative plan, despite having a higher

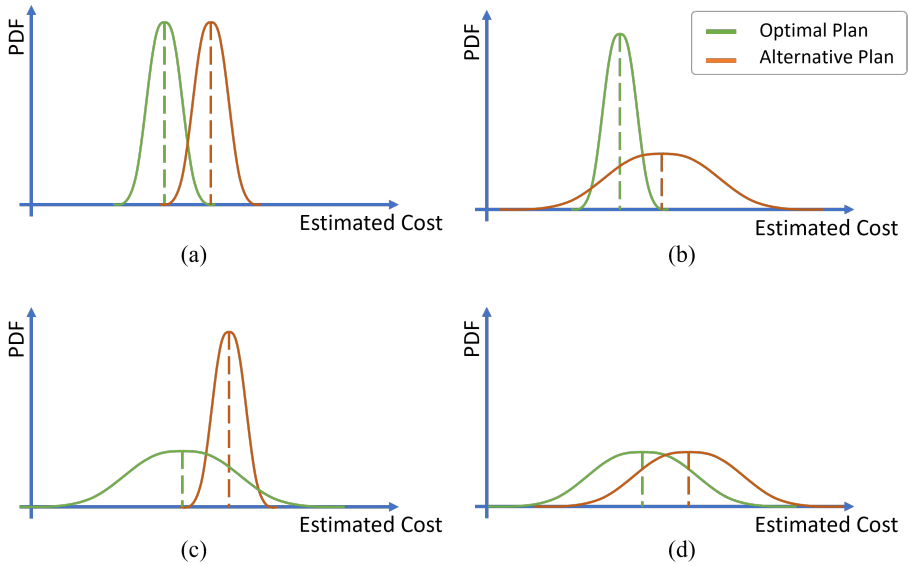


Figure 1.3: Scenarios illustrating plan cost inaccuracies for an expected optimal plan and an alternative (second-best) plan: (a) Cost estimates for both plans are relatively accurate. (b) The cost estimate for the expected optimal plan is more accurate than for the alternative plan. (c) The cost estimate for the expected optimal plan is less accurate than for the alternative plan. (d) The cost estimates for both plans are relatively inaccurate.

expected cost, may still be optimal at runtime with a non-zero probability due to estimation inaccuracies.

1.2.5.2 Scenarios of Cost Inaccuracies

The different subplots in Figure 1.3 represent the following scenarios:

(a) **Accurate Estimates:** In this ideal scenario, the cost estimates for both plans are consistently accurate. The likelihood of the expected optimal plan being more expensive than the alternative plan at runtime is negligible, aligning with the optimizer's assumptions.

(b) **Asymmetric Inaccuracies (Expected Optimal Plan More Accurate):** The cost estimate for the expected optimal plan is more reliable than that of the alternative plan. While this increases confidence in the optimizer's choice, the alternative plan still poses some risk if its estimated cost is underestimated.

(c) **Asymmetric Inaccuracies (Alternative Plan More Accurate):** The cost estimate for the alternative plan is more accurate than that of the expected optimal plan. This increases the likelihood that the optimizer's choice may result in a suboptimal plan at runtime.

(d) **Significant Inaccuracies for Both Plans:** This worst-case scenario arises when both plans have relatively inaccurate cost estimates. In such cases, the risk of suboptimality increases significantly, emphasizing the need for robust query optimization strategies.

1.2.5.3 Implications of Estimation Inaccuracies

The examples depicted in Figure 1.3 underscore the critical importance of considering the *probability distributions* of cost estimates rather than relying solely on expected values. In scenarios with high estimation uncertainties, the optimizer's naive assumptions about cost accuracy can lead to substantial performance degradation.

A more robust approach would involve quantifying the *likelihoods* of suboptimality associated with each plan. Such a method would assess the probability of a plan incurring higher runtime costs compared to its alternatives, accounting for the full cost distribution. The methodology for computing these likelihoods and incorporating them into the optimization process will be discussed in Section 4.3.1.

1.3 Problem Statement

Query optimization involves exploring a large space of candidate execution plans, recursively evaluating a set of plan fragments (sub-plans), selecting the one with the minimum expected cost, and constructing larger plans from smaller fragments, until the creation of complete plan. In comparing plan fragments query optimizers do not consider the uncertainty of cost estimations, as they assume that the costs, as estimated by the cost model, have zero variance. As a result, query optimizers target expected optimality, but not robustness. In the following, the robustness problem is intuitively discussed and a formal definition is provided.

1.3.1 Problem Discussion

Uncertainty of sub-plan evaluation and comparison is traced back to 3 main sources: uncertainty rooted in the (a) plan structure and the data that flow through the plan nodes, (b) limitations of the estimation method, and (c) a crude comparison of a set of plans. We use the terms *risk* and *risky* to point out a high level of *uncertainty* in each of these dimensions. We use the term *robust* if the *sensitivity to uncertainty* is low. Hence, a plan can be considered robust if its execution cost is not sensitive to changes in the parameter

estimates and the environment variables. The level of plan robustness is influenced by its structural characteristics, the operators used in the plan, the patterns of error propagation, and the characteristics of the underlying data. Similarly, an estimation method can be considered robust if it is not sensitive to inaccurate parameter estimates and non-conforming environments. Also, an estimation method should quantify the expected uncertainty for each estimate in order to support robustness. Classic cost models are not robust since they are sensitive to inaccurate parameter estimates and non-conforming environments and do not quantify estimation uncertainty. Due to the uncertainties rooted in the plan structure and data, and the limitations of the cost estimation method, picking a plan from a set of plans is also an uncertain task, as there is a risk that the selected plan will be suboptimal at runtime. We define the sources of uncertainty (or else, risk) in the plan evaluation and selection process as follows:

Definition 1.1. *Plan risk is the uncertainty inherent to the plan, influenced by predicates and operators involved in the plan, the plan structure, error propagation patterns, and the data characteristics.*

Definition 1.2. *Estimation risk is the uncertainty in cost estimates due to limited knowledge of the design and parameters of the ideal cost model. It is influenced by limitations in cost modeling, such as simplifying assumptions and error-prone parameter estimates.*

Definition 1.3. *Suboptimality risk is the likelihood of a plan, selected as optimal from a set of plans, being suboptimal at runtime.*

1.3.2 Problem Formulation

We first formulate the classical problem of optimal plan selection; based on this, we provide two alternative formulations of the problem of robust plan selection. The latter are employed to propose risk-aware plan selection strategies in Section 4.3.

Optimal plan selection problem: Given a finite set of candidate execution plans $\mathcal{P} = (p_1, p_2, \ldots, p_n)$, and a cost function $(f(p_i))$ that estimates the cost of executing a plan $(p_i), i = 1, \ldots, n$, find the plan (p^*) such that: $p^* = \arg\min_{p_i \in \mathcal{P}} f(p_i)$.□

The above problem formulation ignores that estimated costs are inherently inaccurate and that they may lead to selecting plans that are suboptimal at runtime. In contrast, robust query optimization is defined as "an effort to minimize the *sub-optimality risk* by accepting the fact that estimates could be inaccurate [165]." Accordingly, we formulate the problem of robust plan selection as follows:

Robust plan selection problem (Approach 1): Given a finite set of candidate execution plans $\mathcal{P} = (p_1, p_2, \ldots, p_n)$, a cost function $(f(p_i))$ and a robustness function $(g(p_i))$ that estimate the cost and robustness of a plan $(p_i), i = 1, \ldots, n$ respectively,

find the plan (p^*) such that: $p^* = \arg\min_{p_i \in \mathcal{P}} (SOR(p_i))$ where $SOR(p_i)$ is the risk of suboptimality of the plan p_i compared to the alternatives, defined as a function $k(.,.)$ of $f(p_i)$ and $g(p_i)$, i.e. $SOR(p_i) = k(f(p_i), g(p_i))$. $\square$

This approach formalizes the problem as a risk minimization problem, where solving it requires providing approximations for the cost function ($\hat{f}(.)$), the robustness function ($\hat{g}(.)$) and a formulation for the suboptimality risk function ($k(.,.)$). We use this formalization to propose a risk-aware plan selection strategy based on suboptimality risk in Section 4.3.1

Alternatively, we can formalize the problem using an approach that jointly optimizes both cost and robustness. This formalization can be formulated in various ways, such as multi-objective optimization, optimizing a nonlinear combination of the two objectives. Enumerating all such formulations is beyond the scope of this study. Inspired by approaches proposed in the literature [6], we choose to construct a linear combination of cost and robustness functions, as a joint objective function.

The choice of a linear combination is motivated by its intuitive interpretation as a risk-adjusted execution time. When we optimize $p^* = \arg\min_{p_i \in \mathcal{P}} (\alpha \cdot f(p_i) + (1-\alpha) \cdot g(p_i))$, where $f(p_i)$ represents the expected execution time m and $g(p_i)$ represents the standard deviation σ of execution time, this formulation becomes equivalent to optimizing $m + f \cdot \sigma$, where $f = (1-\alpha)/\alpha$ is a scaling factor controlling the weight given to variability. This risk-

adjusted execution time can be interpreted as an estimate of the upper bound of execution time with a certain confidence level. Under the assumption that execution times follow a normal distribution, $m + f \cdot \sigma$ corresponds to the f-sigma upper bound of the execution time. For instance, when $f = 1$, we are optimizing based on a pessimistic cost estimate such that the actual cost would be smaller with a confidence level of approximately 84%. This provides a principled way to balance between average performance and robustness by explicitly accounting for execution time uncertainty.

Robust plan selection problem (Approach 2): Given a finite set of candidate execution plans $\mathcal{P} = (p_1, p_2, \ldots, p_n)$, a cost function $(f(p_i))$ and a robustness function $(g(p_i))$ that estimate the cost and robustness of a plan $(p_i), i = 1, \ldots, n$ respectively, find the plan (p^*) such that: $p^* = \arg\min_{p_i \in \mathcal{P}} (\alpha \cdot f(p_i) + (1 - \alpha) \cdot g(p_i)) \square$

Solving the problem based on this formalization necessitates providing an approximation for the cost function $(\hat{f}(.))$, the robustness function $(\hat{g}(.))$, and a principled approach for determining a suitable weight parameter α. This formalization is used in Section 4.3.2 to propose a conservative plan selection strategy based on risk.

Most of the prior work has been concerned with providing a better approximation for the cost function $(\hat{f}(.))$ [20, 57, 100, 101, 135, 174]. However, providing clear definitions and approximation methods for $\hat{g}(.)$ and $\hat{k}(.,.)$ are missing in the literature. In this thesis, we aim at filling in this gap.

1.4 Contributions

Motivated by the advancements in using ML for query optimization, this work proposes Roq, a novel approach for robust query optimization using approximate probabilistic ML. As such the contributions of the current thesis include:

- Performing a systematic literature review on the concepts of Robustness in the context of query optimization and processing
- Formalizing the notion of plan and cost model robustness to enable an objective evaluation of these characteristics in this work, as well as in future research
- Proposing theoretical methods for quantifying uncertainties and risks in the context of plan evaluation and selection based on approximate probabilistic ML
- Proposing an improved model architecture for a learned cost model that supports measuring uncertainties and provides improved performance compared to the state-of-the-art
- Proposing novel strategies and algorithms for plan evaluation and selection that outperform the state-of-the-art approaches by accounting for uncertainties and risks

- Demonstrating the robustness of the proposed approaches to workload shifts and the limited compilation overheads, which makes them suitable for practical use
- An ablation study on the impacts of accounting for different types of risk in the proposed risk-aware strategies
- Developing RobOpt, a tool for robust workload optimization based on a human-centered approach

1.5 Publications

The following publications are directly derived from the results of this research:

1.5.1 Journal Papers

- Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. Robust plan evaluation based on approximate probabilistic machine learning. Proc. VLDB Endow., 18(8):2626–2638, September 2025. [69]

1.5.2 Conference Papers

- Amin Kamali, Verena Kantere, and Calisto Zuzarte. Robust query optimization in the era of machine learning: State-of-the-art and future directions. In 2024 IEEE 40th International Conference on Data Engineering (ICDE), pages 5371–5375, 2024. [66]

- Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. Robopt: A tool for robust workload optimization based on uncertainty-aware machine learning. In Companion of the 2024 International Conference on Management of Data, SIGMOD '24, page 468–471, New York, NY, USA, 2024. Association for Computing Machinery. [67]

- Baoming Chang, Amin Kamali, and Verena Kantere. A novel technique for query plan representation based on graph neural nets. In Robert Wrembel, Silvia Chiusano, Gabriele Kotsis, A. Min Tjoa, and Ismail Khalil, editors, Big Data Analytics and Knowledge Discovery, pages 299–314, Cham, 2024. Springer Nature Switzerland. [16]

- Chang Liu, Amin Kamali, Verena Kantere, Calisto Zuzarte and Vincent Corvinelli, "A novel framework for join order selection based on reinforcement and representation learning," 2024 IEEE International Conference on Big Data (BigData), Washington, DC, USA, 2024, pp. 8757-8761, doi: 10.1109/BigData62323.2024.10825394. [94]

- Ning Wang, Amin Kamali, Verena Kantere, Calisto Zuzate, Vincent Corvinelli, Brandon Frendo, and Steve Donoghue. A hybrid cost model for evaluating query execution plans. In 2023 IEEE Sixth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pages 133–138, 2023. [149]

1.5.3 Patents

The following patents, filed with the United States Patent and Trademark Office, were direct or indirect derivatives of the research presented in this thesis:

- Kamali, S. M. A., Corvinelli, V., & Zuzarte, C. (2025, April 29). *Learned join cardinality estimation using a join graph representation* (U.S. Patent No. 12287788B2). International Business Machines Corp.

- Kamali, S. M. A., Corvinelli, V., Zuzarte, C., Frendo, B. L., Kantere, V., & Wang, N. (2025, May 15). *Hybrid cost model for evaluating query execution plans* (U.S. Patent No. 20250156417A1). International Business Machines Corp.

- Kamali, S. M. A., Zuzarte, C., Corvinelli, V., Frendo, B. L., Kantere, V., & Wang, N. (2025, May 20). *Robust query execution plan selection using machine learning with predictive uncertainties* (U.S. Patent No. 12306833B2). International Business Machines Corp.

1.5.4 Workshops

In the process of conducting this research, multiple workshops were organized that brought together researchers from the field of using Machine Learning to modernize Database Management Systems. The long abstracts of the workshops are published as follows:

- Jarek Szlichta, Calisto Zuzarte, Verena Kantere, Amin Kamali, Andrew Chai, Xiaohui Yu, Chang Liu, and Baoming Chang. ML4DM '24: The fourth workshop on the emerging applications of machine learning in modern data management. 2024. [139]

- A. Kamali et al., "ML4DM '23: The Third Workshop on the Emerging Applications of Machine Learning in Modern Data Management," in Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering, in CASCON '23. USA: IBM Corp., Nov. 2023, pp. 251–253. [72]

- A. Kamali, J. Szlichta, C. Zuzarte, Y. Chen, V. Kantere, and S. Quader, "ML4DM '22: The second workshop on the emerging applications of machine learning in modern data management," in Proceedings of the 32nd Annual International Conference on Computer Science and Software Engineering, in CASCON '22. USA: IBM Corp., Nov. 2022, pp. 229–231. [70]

- A. Kamali, C. Zuzarte, and V. Kantere, "Emerging applications of machine learn-

ing in modern data management," in Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering, in CASCON '21. USA: IBM Corp., Nov. 2021, pp. 284–286. [71]

1.6 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 presents the Design Science Research methodology that guides this investigation, detailing the research philosophy, DSR framework application, artifact development approach, and evaluation strategy. Chapter 3 presents a comprehensive literature review on robustness in query optimization and processing, as well as machine learning approaches with predictive uncertainty. It examines various definitions of robustness, existing methods for robust plan evaluation, and techniques for uncertainty quantification in machine learning models. Chapter 4 introduces the theoretical framework for robust plan evaluation based on approximate probabilistic machine learning, including the formalization of plan and cost estimation uncertainty modeling and the proposed approach for ensuring robustness via risk quantification. Chapter 5 details the model architecture of the risk-aware learned cost model, describing the query and plan encoding schemes, representation learning methods, and estimation modules. Chapter 6 provides a comprehensive experimental study evaluating the proposed approaches against

state-of-the-art techniques using various benchmarks and performance metrics. Chapter 7 presents RobOpt, a tool for robust workload optimization based on uncertainty-aware machine learning, explaining its architecture, component modules, and use case scenarios. Finally, Chapter 8 concludes the thesis by summarizing the contributions and discussing potential directions for future work.

Chapter 2

Methodology

This chapter establishes the methodological foundation for this research by presenting the Design Science Research (DSR) paradigm that guides the investigation of robust query optimization in relational databases. Section 2.1 introduces the purpose and scope of the methodology chapter. Section 2.2 examines the philosophical underpinnings of the research approach, while Section 2.3 provides an overview of the Design Science Research framework adopted for this study. Section 2.4 details the application of the DSR process model to this research, including problem identification, objectives definition, design and development, demonstration, evaluation, and communication. Section 2.5 describes the specific artifacts developed in this research. Section 2.6 outlines the evaluation methods used to assess the effectiveness of the proposed artifacts. The chapter concludes with Section 2.8, which

summarizes the methodological choices and their alignment with the research objectives.

2.1 Introduction

The selection of an appropriate research methodology is fundamental to ensuring the rigor, relevance, and validity of academic research. This chapter presents the methodological framework that guides this investigation into robust query optimization in relational databases based on approximate probabilistic machine learning. The research addresses both theoretical challenges in understanding robustness in database systems and practical problems faced by database developers, administrators, and end-users in real-world deployment scenarios.

Given the nature of this research problem—which involves both the development of novel theoretical frameworks and the creation of practical artifacts (cost models, optimization algorithms, and tools)—Design Science Research (DSR) emerges as the most appropriate methodological paradigm. DSR is particularly well-suited for research that aims to create and evaluate artifacts designed to solve identified organizational problems [56]. This methodology bridges the gap between theoretical knowledge and practical application, ensuring that the research contributes both to the academic understanding of robust query optimization and to the practical advancement of database management systems.

The choice of DSR is further motivated by the multidisciplinary nature of this research, which draws from database systems, machine learning, and uncertainty quantification domains. DSR provides a structured approach to synthesize knowledge from these diverse fields while creating novel artifacts that address the identified research problem. The methodology ensures that the research maintains both academic rigor through systematic evaluation and practical relevance through the development of implementable solutions.

2.2 Research Philosophy and Approach

2.2.1 Philosophical Foundations

This research is grounded in a pragmatic philosophical stance, which aligns naturally with the DSR paradigm. Pragmatism emphasizes the practical consequences of research and the utility of knowledge in solving real-world problems [99]. From an epistemological perspective, this research adopts a pluralistic approach, recognizing that knowledge can be generated through multiple means including both empirical observation and constructive design activities.

The ontological position acknowledges that while objective realities exist in database systems (such as measurable query execution times and system performance metrics), the

perception and interpretation of "robustness" may vary among different stakeholders. This perspective is consistent with the human-centered approach adopted in this research, which considers the needs of database developers, administrators, and end-users as equally valid but potentially different viewpoints on system robustness.

2.2.2 Research Approach

This research primarily follows an abductive reasoning approach, which is characteristic of DSR [122]. Abductive reasoning involves iterating between observation of practical problems, theoretical understanding, and solution design. The research begins with observations of real-world problems in query optimization (performance unpredictability, sensitivity to estimation errors), draws upon existing theories from database systems and machine learning, and creates novel artifacts to address these problems.

The abductive approach manifests through iterative cycles of problem observation, theory building, artifact creation, and evaluation-based refinement, ensuring that theoretical insights are grounded in practical applicability.

2.3 Design Science Research Framework

2.3.1 DSR Definition and Relevance

Design Science Research is defined as a research paradigm that seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts [56]. In the context of information systems, DSR focuses on the creation and evaluation of artifacts such as constructs, models, methods, and instantiations that address important and relevant business problems.

For this research, DSR is particularly relevant because it addresses the fundamental challenge of translating theoretical insights about robustness in query optimization into practical solutions that can be implemented in real database systems. The research problem inherently requires both understanding the theoretical foundations of robustness and uncertainty quantification and creating tangible artifacts that demonstrate the practical applicability of these concepts.

2.3.2 DSR Guidelines

This research adheres to the key DSR guidelines proposed by Hevner et al. [56], creating multiple artifacts (theoretical frameworks, cost model architectures, optimization

algorithms, and software tools) that address significant problems in database management systems. The research provides rigorous evaluation through experimental studies and comparative analysis, contributes novel design and descriptive knowledge, and follows an iterative refinement process based on evaluation results.

2.3.3 Knowledge Contribution Types

Following the framework of Gregor and Hevner [46], this research contributes across multiple knowledge levels, from practical instantiations (RobOpt tool) to novel theoretical inventions (risk-aware query plan selection framework).

2.4 DSR Process Model Application

This research follows the six-activity DSR methodology (DSRM) proposed by Peffers et al. [122]. This process model provides a structured approach to conducting DSR while maintaining flexibility in execution. The following subsections detail how each activity is applied in this research.

2.4.1 Activity 1: Problem Identification and Motivation

The research problem was identified through a comprehensive analysis of stakeholder needs in database management systems, as detailed in Chapter 1. Three distinct perspectives were examined: database developers facing high engineering costs in optimizer development, database administrators dealing with operational overhead in system tuning, and end-users requiring predictable system performance.

The problem identification process revealed that existing query optimization approaches primarily focus on average-case performance while neglecting worst-case scenarios and performance predictability. This gap motivates the need for robust query optimization approaches that can provide performance guarantees and minimize the risk of significant performance degradation.

The motivation for this research is further strengthened by the observation that recent machine learning approaches to query optimization, while promising in their ability to improve average performance, introduce new sources of uncertainty that are not systematically addressed in current literature.

2.4.2 Activity 2: Objectives of a Solution

Based on the problem analysis and stakeholder needs identified in Chapter 1, the research objectives are formulated to address the specific requirements of database developers, administrators, and end-users:

1. **Theoretical Objective**: Develop a comprehensive theoretical framework for understanding and quantifying robustness in query optimization that accounts for multiple sources of uncertainty

2. **Methodological Objective**: Create practical methods for quantifying plan risk, estimation risk, and suboptimality risk in ML-based query optimization systems

3. **Algorithmic Objective**: Design and implement risk-aware plan selection strategies that balance average performance with robustness guarantees

4. **Architectural Objective**: Develop an uncertainty-aware cost model architecture that provides both cost estimates and confidence measures

5. **Tool Objective**: Create a practical tool for robust workload optimization that addresses the needs of different user personas

6. **Evaluation Objective**: Demonstrate effectiveness through comprehensive experimental evaluation across multiple benchmarks representing diverse stakeholder scenarios

These objectives collectively aim to advance both theoretical understanding and practical implementation of robust query optimization while addressing the specific challenges faced by each stakeholder group.

2.4.3 Activity 3: Design and Development

The design and development activity involves creating multiple interconnected artifacts that address the research objectives:

Theoretical Framework Development The theoretical framework development follows a systematic approach:

- **Literature Synthesis**: Comprehensive review of robustness definitions and uncertainty quantification methods across database systems and machine learning domains
- **Formal Modeling**: Mathematical formalization of uncertainty decomposition in cost estimation using the law of total variance

- **Risk Quantification**: Development of measures for plan risk, estimation risk, and suboptimality risk specifically tailored for ML-based cost models

Architectural Design The cost model architecture is designed with the following principles:

- **Modularity**: Separation of uncertainty quantification from cost estimation to enable the extension of any learning-based cost models with uncertainty quantification
- **Scalability**: Support for complex join graphs and query execution plans
- **Interpretability**: Provision of uncertainty estimates that can be understood and acted upon by database administrators

Algorithm Development Risk-aware optimization algorithms are developed through:

- **Strategy Formulation**: Development of conservative and suboptimality-based plan selection strategies
- **Implementation**: Translation of theoretical strategies into practical algorithms
- **Integration**: Incorporation of algorithms into existing query optimization frameworks

2.4.4 Activity 4: Demonstration

The demonstration activity involves showing the practical applicability of the developed artifacts through benchmark evaluation on JOB, CEB, DSB, and TPC-DS+, development of the RobOpt tool that addresses stakeholder needs identified in Chapter 1, examination of critical scenarios such as workload shifts, and comparative analysis demonstrating improvements over existing approaches.

2.4.5 Activity 5: Evaluation

The evaluation strategy follows the Framework for Evaluation in Design Science Research (FEDS) [146], incorporating multiple evaluation methods to ensure comprehensive assessment:

The evaluation strategy combines controlled experiments using both synthetic and real-world benchmarks, theoretical analysis of algorithm properties, and empirical measurement of performance improvements. Multiple criteria assess artifact effectiveness including efficacy, efficiency, robustness, and practical usability. Specific methods include experimental studies, comparative analysis with state-of-the-art approaches, ablation studies, and sensitivity analysis for parameter variations and workload changes.

2.4.6 Activity 6: Communication

The communication activity ensures that research findings reach relevant academic and practitioner audiences through peer-reviewed publications, open-source tool releases, tutorial presentations, workshops, technical reports, and patent applications, as detailed in Section 1.5.

2.5 Artifact Development

This research develops multiple interconnected artifacts that collectively address the research problem. The artifacts are designed to be both theoretically sound and practically implementable.

2.5.1 Primary Artifacts

Theoretical Framework (Chapter 4)

- **Uncertainty Decomposition Model**: Mathematical framework using the law of total variance to separate plan risk from estimation risk
- **Risk Quantification Measures**: Formal definitions and computation methods for

plan risk, estimation risk, and suboptimality risk using approximate probabilistic ML

- **Risk-Aware Plan Selection Strategies**: Conservative plan selection, suboptimality risk minimization, and search space pruning algorithms

Roq Architecture (Chapter 5)

- **Query Encoding Module**: Graph Neural Network-based architecture using extended TransformerConv for learning query representations from join graphs
- **Plan Encoding Module**: Tree-based encoding of query execution plans with operator and table participation information
- **Representation Learning**: Query Processor (GNN) and Plan Processor (TCNN) for learning embeddings
- **Dual-Branch Estimation Module**: Simultaneous prediction of execution time and uncertainty using MC Dropout and heteroscedastic modeling

RobOpt Tool (Chapter 7)

- **RLCM Trainer**: Component for training risk-aware learned cost models
- **Strategy Optimizer**: Component for tuning risk-aware strategy parameters

- **Workload Optimizer**: Component for determining optimal strategies per query
- **Strategy Analyzer**: Component for comparing strategy performance and risk profiles
- **Workload Analyzer**: Component for identifying problematic queries and risk patterns
- **Workload Planner**: Component for automated strategy selection using query embeddings

2.5.2 Artifact Design Principles

The artifacts are designed according to the following principles:

1. **Modularity**: Each artifact can be developed, tested, and deployed independently
2. **Extensibility**: Artifacts can be extended to incorporate new uncertainty quantification methods or optimization strategies
3. **Interoperability**: Artifacts can integrate with existing database systems and optimization frameworks
4. **Scalability**: Artifacts can handle realistic workloads and database sizes

5. **Interpretability**: Artifacts provide explainable outputs that can guide decision-making

2.6 Evaluation Strategy

The evaluation strategy is designed to comprehensively assess the effectiveness, efficiency, and practical applicability of the developed artifacts. Following DSR evaluation principles [56, 146], the evaluation employs multiple methods and criteria to ensure rigor and relevance.

2.6.1 Evaluation Framework

The evaluation framework is structured around three key dimensions:

1. **Technical Effectiveness**: Assessment of whether the artifacts solve the intended technical problems

2. **Practical Utility**: Evaluation of the artifacts' usefulness in realistic database environments

3. **Stakeholder Impact**: Analysis of how the artifacts address the needs of different stakeholder groups

2.6.2 Evaluation Methods

Experimental Evaluation Controlled experiments are conducted using four benchmarks:

- **Join Order Benchmark (JOB)**: 113 queries with up to 27 joins over IMDB dataset, evaluated using 10-fold cross-validation
- **Cardinality Estimation Benchmark (CEB)**: 1000 queries with up to 12 joins over IMDB dataset for workload shift analysis
- **Decision Support Benchmark (DSB)**: 1000 queries based on enhanced TPC-DS with realistic data distributions
- **TPC-DS+**: 13,000 synthetic queries with up to 4 joins, including many-to-many and cyclic joins

Comparative Analysis Systematic comparison with existing approaches:

- **Traditional Baselines**: IBM Db2 optimizer and analytical cost model
- **ML-based Baselines**: Neo, Bao, Lero, and Balsa learned cost models
- **Ablation Studies**: Analysis of individual risk components and strategy effectiveness

Robustness Analysis Specific evaluation of robustness characteristics:

- **Workload Shift Testing**: Cross-benchmark evaluation (training on one benchmark, testing on another)
- **Suboptimality Analysis**: Measurement of worst-case performance using percentile metrics
- **Uncertainty Quantification**: Evaluation of plan risk, estimation risk, and suboptimality risk measures

2.6.3 Evaluation Metrics

Evaluation Metrics

- **Prediction Accuracy**: Q-error defined as $Q_{error} = \frac{\max(y,\hat{y})}{\min(y,\hat{y})}$ where y is actual and $\hat{y}$ is predicted execution time
- **Correlation**: Spearman's rank correlation coefficient between predicted and actual execution times (more suitable than Pearson's for ranking-based plan selection)
- **Plan Suboptimality**: $\text{Subopt}(p_i) = \frac{ET(p_i)}{ET(p^*)}$ where p^* is the best performing plan, with distribution analysis focusing on tail-end performance (median, mean, 95th, 99th percentiles, max)

- **Runtime Performance**: Total workload execution time to ensure robust optimization doesn't cause major slowdowns
- **Inference Overheads**: Inference time overhead for risk-aware plan evaluation algorithms, particularly for suboptimality risk computation

2.7 Ethical Considerations

This research involves technical development and evaluation of database optimization methods using publicly available benchmarks and synthetic data. The primary ethical considerations include:

- **Data Usage**: All experimental evaluation uses publicly available benchmark datasets (IMDB, TPC-DS) or synthetic data, ensuring no privacy concerns
- **Reproducibility**: Core artifacts and experimental frameworks are made available as open-source software to enable reproducibility
- **Academic Integrity**: Proper attribution of prior work and clear delineation of novel contributions, with appropriate handling of industry collaborations
- **Limitation Disclosure**: Clear communication of approach limitations and potential failure modes in experimental results

2.8 Conclusion

This methodology chapter has established the Design Science Research framework that guides this investigation into robust query optimization in relational databases. The DSR paradigm is well-suited to this research because it addresses both theoretical challenges in understanding robustness and uncertainty quantification and practical needs for implementable solutions in database management systems.

The DSR process model provides a structured approach to developing and evaluating multiple interconnected artifacts including theoretical frameworks (Chapter 4), architectural designs (Chapter 5), optimization algorithms, and practical tools (Chapter 7). The evaluation strategy ensures comprehensive assessment through experimental studies across multiple benchmarks, addressing the specific needs of database developers, administrators, and end-users identified in Chapter 1.

The methodological choices reflect a commitment to both academic rigor and practical relevance. The abductive reasoning approach enables iterative refinement between theoretical understanding and practical implementation, while the comprehensive evaluation strategy ensures meaningful contributions to both academic knowledge and industry practice.

This methodological foundation supports the subsequent chapters, which detail the theoretical framework, architectural design, experimental evaluation, and tool development that collectively demonstrate the practical value of robust query optimization based on approximate probabilistic machine learning.

Chapter 3

Literature Review

This comprehensive literature review chapter establishes the theoretical and methodological foundation for research on robustness in query optimization. Section 3.1.1 explores various definitions of robustness in database systems through three conceptual frameworks: trade-off between predictability and performance, low sensitivity to unexpected conditions, and worst-case performance guarantees. The chapter then provides an overview of selected studies (Section 3.1.2) before systematically analyzing existing research through a multi-level taxonomy, covering robust operators (Section 3.1.3) with adaptive access and join implementations, approaches to quantifying plan robustness (Section 3.1.4) using PCF derivatives and various suboptimality measures, innovations in robust statistics (Section 3.1.5) for improved cardinality estimation, advancements in cost models that account for

uncertainty (Section 3.1.6), search algorithms that balance exploration and exploitation (Section 3.1.7), and execution techniques that adapt at runtime (Section 3.1.8). Section 3.2 delves into machine learning with predictive uncertainty, beginning with general uncertainty modeling approaches (Section 3.2.1) before distinguishing between aleatoric or data uncertainty (Section 3.2.2) and epistemic or model uncertainty (Section 3.2.3), while reviewing techniques for quantifying these uncertainties in neural networks. By synthesizing insights from both database systems and machine learning literature, this chapter identifies research gaps and establishes the conceptual foundation for a novel approach to robust query optimization using approximate probabilistic machine learning.

3.1 Robustness in Query Optimization and Processing

Running queries in database management systems typically involves accessing, joining, and aggregating data from different sources. Such queries can be executed in several different ways determined by a query execution plan. The difference in the performance of these plans can be several orders of magnitude large. Therefore, sophisticated query optimizers are developed to search for the best-performing plan at compile-time. Query optimizers rely on a limited number of search algorithms to traverse a search space of plans and find the one

that is expected to have optimal performance. To this end, they rely on a cost model that is used to evaluate each plan in the search space to find the one with minimum expected cost, a.k.a. the "optimal plan". The cost model takes as input some parameters, most notably the size of the data that flows through the plan operators, also known as cardinalities. The values of these parameters are typically unknown at compile time. Therefore, the optimizer relies on their estimates. Due to limitations in the cardinality estimation methods, these estimates are inherently subject to arbitrary errors. In addition, the cost model is designed by making certain assumptions that may not hold in practice. For these reasons, the plan selected by the optimizer may end up being suboptimal at runtime. In other words, the selected plan may not be robust to estimation errors and to environments where simplifying assumptions do not hold. Therefore, query optimization approaches that are less sensitive to estimation errors and do not rely on simplifying assumptions may be considered more robust. It is important to note that while query optimization has other phases such as query rewrite, which precedes the plan selection phase, such aspects are beyond the scope of this thesis.

3.1.1 Robustness Definitions

Despite the high importance of robustness for database systems and decades of studies on the subject, there does not appear to be a consensus on the definition of robustness [52].

Therefore, in this section, an overview of how the concept is defined in various studies is provided, with the goal of establishing a comprehensive understanding of the concept.

3.1.1.1 Trade-off Between Predictability and Performance

Some studies suggested *stability*, *consistency*, and *predictability* as the main characteristics of a robust system [6, 24, 44]. Chu et al. suggested that providing good performance on average is not enough [24]. They argued that systems that perform consistently suboptimal by an acceptable factor are superior to ones that perform fast on average but fail miserably in exceptional situations. Babcock et al. [6] argued that database systems are typically components of larger systems involving many more applications. Therefore, "tuning and testing of the system as a whole is greatly simplified when its components behave predictably [6]". Graefe et al. [44] argued that consistently slow query performance by a moderate factor can be compensated for by cheap concurrency and parallel computing techniques while unpredictably fast database systems require expensive experts to race to isolate, understand, and address individual problems as they arise [44]. These studies also acknowledge that predictability can sometimes be at odds with raw performance, while they are both desirable properties of a database system. Therefore, *striking a trade-off between these two objectives would become the goal of a robust system.* For example, Chu et al. suggested the runtime *variance* as a measure of plan stability and proposed min-

imizing a linear combination of runtime and its variance as a way to achieve a balance between performance and robustness [24]. As another example, Babcock et al. [6] targeted the same objective based on the probability density function of the cost. Instead of taking the expected cost, they took the T^{th} percentile of the cost cumulative density function $cdf(C)$ as the point of comparison: $cdf^{-1}(T\%)$. The "confidence threshold" T implies that there is $T\%$ chance that the cost of the plan is less than $cdf^{-1}(T\%)$. Using higher values for T leads to picking more predictable plans over plans that are faster on average but have highly suboptimal worst-case performance.

Based on these approaches, robustness can be defined and quantified as follows:

Definition 3.1. *In query optimization,* ***robustness*** *is the ability of a database system to sustain stable and predictable performance across varying or unexpected conditions by favoring consistent plan reliability over peak speed, thus avoiding severe latency or highly suboptimal outcomes.*

Quantification: *Robustness is measured by a combination of runtime variance and average cost, or by a pessimistic estimate of the plan's cost (e.g., the T^{th} percentile of the cost distribution).*

3.1.1.2 Low Sensitivity to Unexpected Conditions

Chu et al. [24, 25] showed that unexpected conditions at run-time arising from inaccurate cardinality estimates or resource contentions can adversely affect the selection of a good plan. They proposed choosing the plan with the "Least Expected Cost" based on the probability distribution of the uncertain parameters. Based on these observations, Harish et al. [51] defined *robust plans* as ones that are relatively less sensitive to selectivity estimation errors. Graefe et al. [44] built upon these works and defined *robustness as the ability of the system to handle unexpected conditions, including both uncertain parameters and available computing resources*. In the context of selecting an optimal plan, this sensitivity is characterized by how gracefully the performance of the plan degrades in the presence of uncertain parameters or inadequate resources. This definition or its variations are extensively adopted in the literature [165]. In extreme cases, this sensitivity characterizes itself as a *performance cliff* [52]. Performance cliffs are cases where the performance abruptly degrades with slight changes in parameters, workload, or available resources. Consequently, it can be argued that *a robust system must not have performance cliffs*.

From this perspective, robustness can be defined and quantified as follows:

Definition 3.2. *In query optimization,* ***robustness*** *is defined as the ability of a database system to maintain stable performance with low sensitivity to unexpected conditions, such*

as estimation errors or resource variations, ensuring graceful degradation without abrupt performance drops.

***Quantification**: Robustness is quantified by the "Expected Cost" over uncertain conditions or by the plan's sensitivity to parameter variations, keeping system performance within predictable bounds despite imperfect information.*

3.1.1.3 Worst-case Performance Guarantees

Moerkotte et al. were the first to establish guarantees for performance degradation with respect to the size of cardinality misestimations measured by the multiplicative error (q-error) [108]. They showed that bounding the impact of cardinality misestimation on the base relation to q guarantees selecting a plan that is at most q^4 more costly than the optimal plan. A few years later, Dutt et al. [31] suggested bounded worst-case performance as a measure of robustness. They argued that robustness is the ability to provide provable guarantees for the worst-case performance compared to the optimal performance. Therefore, they proposed Maximum Suboptimality (MSO) as a measure of robustness. Later, this definition and the corresponding measure were used by several other studies [32, 75–77, 77].

From this perspective, robustness can be defined and quantified as follows:

Definition 3.3. *In query optimization, **robustness** is the ability of a system to guarantee*

that performance remains within a bounded factor of the optimal, even under worst-case scenarios like estimation errors or resource contentions.

***Quantification**: These guarantees are quantified through metrics such as Maximum Suboptimality (MSO), ensuring predictable performance regardless of uncertainties.*

3.1.2 Overview of Selected Studies

In the context of query optimization and processing, robustness is studied from different perspectives. We propose a multilevel taxonomy to effectively classify the state-of-the-art with respect to common characteristics of the proposed approaches in each category. At the highest level, robustness studies can be classified based on the components they aim to improve [52], including: 1) Robust operators, 2) Robust plans, 3) Robust statistics, 4) Robust plan evaluation, 5) Robust search algorithms, and 6) Robust execution. These classes can be further broken down on the basis of the approach the studies take to improve robustness, as illustrated in Figure 3.1.

In the following sections, these approaches to robustness are introduced and the selected respective studies are reviewed in detail. This is followed by an analysis of their advantages and limitations in achieving robustness.

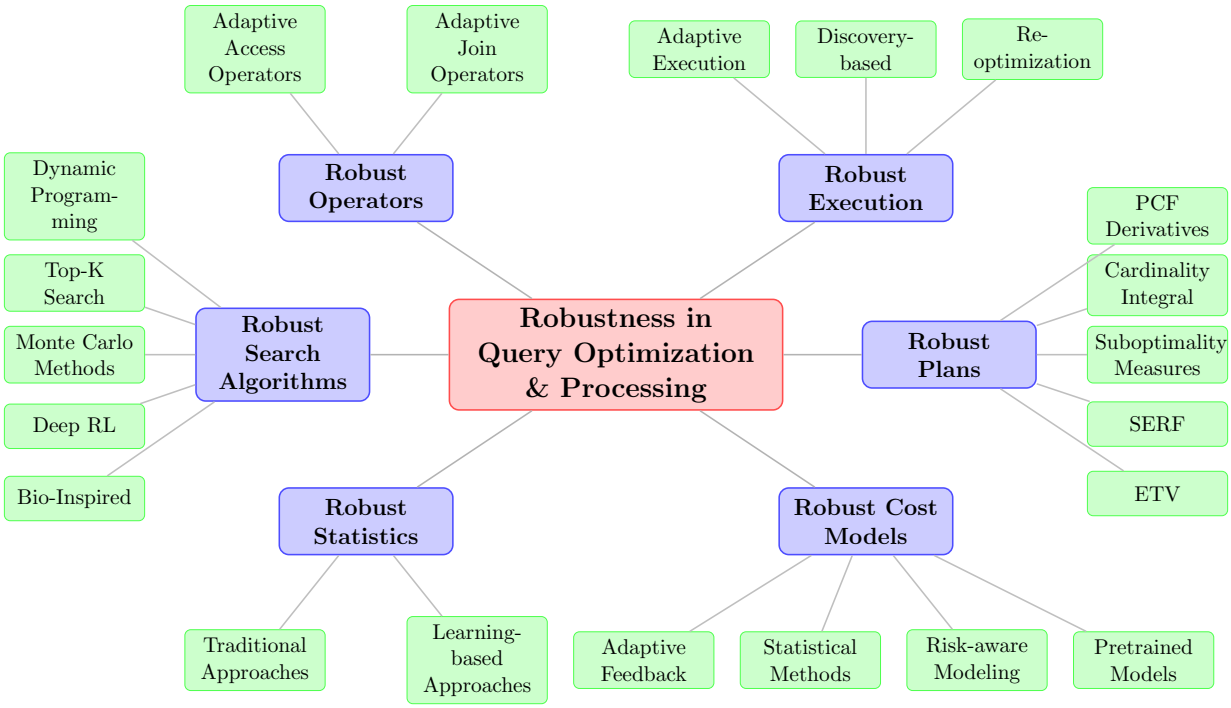


Figure 3.1: Taxonomy of robustness approaches in query optimization and processing, showing the six main categories and their respective subcategories

3.1.3 Robust Operators

At the most granular level, the robustness of query optimization and processing can be analyzed at the plan operator level. This involves examining how various access and join operators perform under varying parameter values. The following subsections review studies that specifically address the robustness of access [12] and join operators [45,118,174].

3.1.3.1 Adaptive Access Operators

Some studies have proposed new access operators that are less sensitive to cardinality estimate inaccuracies, to tackle the problem of inaccurate statistics leading to suboptimal query plans. For example, Smooth Scan [12] introduces a novel access path operator that dynamically adapts its behaviour during query execution based on the observed data distribution, eliminating the reliance on potentially outdated or unavailable statistics. Smooth Scan seamlessly transitions between an index scan at low selectivities and a full table scan at high selectivities, achieving near-optimal performance across a wide range of selectivity values. This adaptive nature makes Smooth Scan highly robust to cardinality estimation errors, ensuring consistent performance regardless of the accuracy of statistics. The authors implement Smooth Scan in PostgreSQL and present a detailed cost model to evaluate its effectiveness. Experimental results demonstrate that Smooth Scan consistently outper-

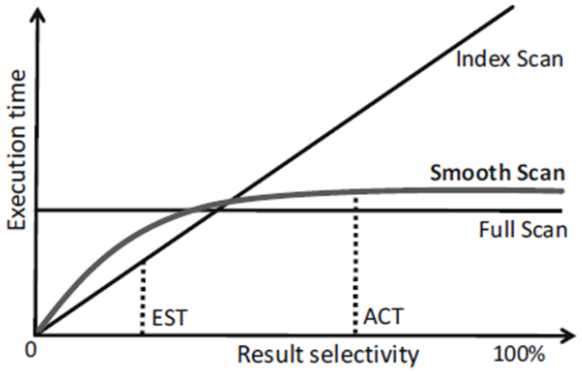


Figure 3.2: The target behavior of Smooth Scan [12]

forms traditional access paths, particularly when statistics are inaccurate, highlighting its potential for robust and efficient query processing.

3.1.3.2 Adaptive Join Operators

Another group of studies suggest new join algorithms that are less sensitive to cardinality estimation errors. For example, Adaptive Join Algorithm (AJA) [174] is designed to switch between a Hash Join algorithm and a Nested Loop Join algorithm at runtime, depending on which one is estimated to be more efficient based on the actual data characteristics observed during execution. This adaptability stems from the recognition that the optimizer's initial choice of join algorithm might be based on inaccurate cardinality estimations, leading to performance degradation if the chosen algorithm is not suitable for the actual data distribution.

The Generic Join Algorithm (GJA) [118], on the other hand, is a two-phase process designed to achieve worst-case optimal performance for join operations. In the preparation phase, a compatible total ordering of attributes is determined, and search trees or hash indices are constructed for all relations, ensuring consistency with this ordering. The join phase then uses a recursive procedure, prioritising the processing of "light" tuples (those with low fanout) to control intermediate result sizes. This careful selection of join order and tuple processing, guided by per-tuple cardinality estimations, ensures adherence to the AGM bound, a theoretical upper limit on the join result size. This approach guarantees optimal performance in the worst-case scenario, regardless of data distribution or join order, though it may introduce an index overhead and its performance on real-world datasets may vary.

3.1.3.3 Merits of Robust Plan Operators

Robust plan operators are designed to handle the uncertainties inherent in query processing, specifically errors in cardinality estimation. Here are some general merits of these operators:

Mitigation of Estimation Errors: These approaches primarily address the challenge of inaccurate cardinality and selectivity estimations, which often lead to sub-optimal plan choices and performance degradation. They aim to minimize the impact of these

errors on query execution time and provide more predictable performance [12, 45, 174].

Performance Stability: They aim to provide consistent performance across a wide range of data distributions and selectivities, reducing the risk of "performance cliffs" caused by inaccurate estimations. This stability makes them particularly beneficial for applications demanding predictable query response times [52, 178].

Reduced Optimization Overhead: Robust operators adapt their behavior based on runtime observations, minimizing the reliance on upfront cardinality estimations [12, 174] or join ordering [118] and potentially reducing the need for complex statistics gathering and maintenance.

3.1.3.4 Limitations of Robust Plan Operators

Despite their advantages, robust plan operators also have limitations:

Complexity: Implementing robust operators often involves significant modifications to existing database systems and can be algorithmically complex [52], as seen with the GJA which necessitates specialized index structures and potentially substantial system-level changes [118]. In general, implementing these operators in database systems necessitates substantial changes to the query optimizer, the cost model, and the execution engine. Furthermore, they introduce new risks given their unknown interactions with the other

components of a database system.

Overhead: The pursuit of robustness can sometimes introduce overhead. For instance, the index structures required by the GJA [118], while enabling worst-case optimality, can lead to high storage overhead and increased preparation time. As another example, Smooth Scan [12] requires several auxiliary data structures to function efficiently.

Potential Inefficiency in Average Cases: Operators optimized for worst-case scenarios, like the GJA [118], may not be the most efficient choice for average cases where simpler, estimation-based algorithms could perform better [52].

Limited Scope of Applicability: Not all robust plan operator approaches are universally applicable. Some techniques might be tailored for specific operator types [12, 174] or query structures [118] limiting their generalization.

While these approaches share some characteristics, they have substantial differences too. While Smooth Scan targets access operators, AJA and GJA target join operators. Furthermore, AJA and GJA have substantial differences too. Table 3.1 summarizes a comparison between AJA and GJA, in terms of similarities, difference, pros, and cons.

Table 3.1: Comparison of Adaptive Join Algorithm and Generic Join Algorithm

Aspects	**Adaptive Join Algorithm (AJA)** [174]	**Generic Join Algorithm (GJA)** [118]
Similarities	Uses runtime information for join optimization.	Uses runtime information (per-tuple cardinality estimation) for join optimization.
Differences	**Scope:** Adapts the join algorithm for individual join operations. **Index Reliance:** Does not rely on specialized indices. **Theoretical Guarantees:** Lacks theoretical guarantees on worst-case performance.	**Scope:** Aims for worst-case optimality for the entire join query, controlling intermediate result sizes. **Index Reliance:** Heavily relies on precomputed indices for prefix lookups. **Theoretical Guarantees:** Adheres to the AGM bound, guaranteeing worst-case optimality.
Advantages	**Simplicity:** Easy to implement, minimal modifications to existing systems. **Low Overhead:** No need for specialized indices, lower storage and preparation time. **Adaptability:** Adapts to various data distributions and query characteristics.	**Worst-Case Optimality:** Guarantees bounded runtime complexity regardless of data distribution or join order. **Robustness:** Performs consistently well even in challenging scenarios due to AGM bound adherence. **Suitability for Specific Scenarios:** Ideal for real-time systems or applications requiring predictable query times.
Disadvantages	**Lack of Theoretical Guarantees:** Performance can vary significantly depending on the data. **Limited Scope:** Only adapts individual joins, not the entire query.	**Index Overhead:** High storage overhead and increased preparation time due to index construction. **Complexity:** Difficult to implement, requires substantial modifications to existing systems. **Potential Inefficiency:** May not be the most efficient option for average cases. **Limited Applicability:** Only applicable to left-deep plans

3.1.4 Robust Plans

Plan robustness refers to a plan's characteristics determined by its cost behavior across various parameter values. To quantify and compare the robustness of different query execution plans, researchers have developed several measures that capture different aspects of plan stability and performance predictability. These measures are typically based on the Parametric Cost Function (PCF), which defines the value of the cost function at various points in the parameter space [155]. The parameter space encompasses all possible combinations of values for error-prone parameters that influence the plan's cost, with cardinality estimates being particularly notable.

The following subsections examine the primary measures used to assess plan robustness, each addressing different characteristics of robust query execution. These include derivative-based measures that assess sensitivity to parameter changes, integral-based measures that evaluate overall cost behavior across the parameter space, suboptimality measures that quantify worst-case performance degradation, and specialized metrics that capture specific aspects of plan volatility and error resistance.

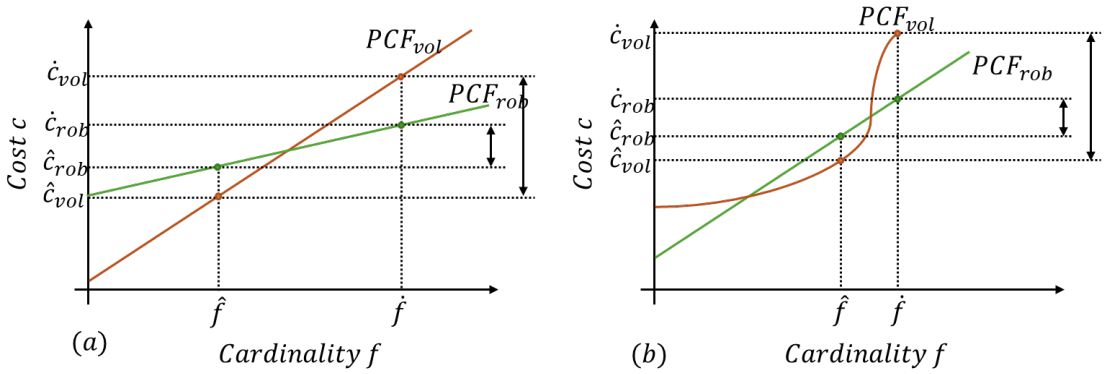


Figure 3.3: The PCF of two plans with respect to a cardinality value highlighting (a) the role of the first derivative in determining graceful performance degradation, and (b) the role of the second derivative of PCF in identifying a performance cliff

3.1.4.1 The First and the Second Derivative of PCF

The sensitivity of a plan's cost to changes in the parameter estimates characterizes its volatility. This means that with slight changes in the parameters, the cost of a volatile plan can increase drastically. This sensitivity can be quantified using the Parametric Cost Function (PCF). Figure 3.3-a illustrates PCFs for two plans with respect to a single error-prone cardinality value, assuming that the cost grows linearly with respect to cardinality. $\dot{f}$ represents the true cardinality, while $\hat{f}$ represents its estimated value at compile time.

Note that in this case, the optimizer will pick the plan represented by PCF_{vol} given its lower estimated cost $\hat{c}_{\text{vol}}$, while its true cost $\dot{c}_{\text{vol}}$, given the true cardinality, is higher than that of PCF_{rob}. In this example, the plan represented by PCF_{rob} is more *robust* in comparison to the plan represented by PCF_{vol}, which is *more volatile* or *riskier*.

To make the concept more concrete, a PCF (and hence the plan it represents) has *graceful performance degradation* if its first derivative (i.e., slope) in the vicinity of the parameter value is small. This means that in the presence of an error, the plan's cost does not change abruptly. In the example illustrated in Figure 3.3-a, PCF_{rob} represents a plan with more graceful performance degradation compared to the plan represented by PCF_{vol}.

Some methods for quantifying the robustness of a query plan based on the first derivative of the PCF have been previously proposed [154] as the Cardinality-Slope (CS) robustness metric. Calculated for each edge in the plan, Cardinality-Slope considers the slope of the PCF at the estimated cardinality and an edge weighting function that accounts for the sensitivity of different operators to cardinality errors [154]. Summing the weighted slopes yields the plan's overall robustness.

Using the first derivative as a robustness criterion, however, is valid only if the PCF is assumed to be linear. In practice, this assumption does not hold. The notion of a *performance cliff* should also be defined to account for non-linear changes. In mathematical terms, a PCF does not have a performance cliff if it exhibits graceful performance degradation and if its second derivative (i.e., concavity) in the vicinity of the parameter value is not large. This means that in the presence of an error, the plan's cost does not accelerate.

In the example illustrated in Figure 3.3-b, the plan represented by PCF_{vol} has a lower estimated cost $\hat{c}_{\text{vol}}$ at the estimated cardinality $\hat{f}$. In addition, its slope at $\hat{f}$ is not larger than

that of PCF_{rob}. However, it still cannot be considered robust since its second derivative is larger than the other plan. In other words, in the vicinity of the estimated cardinality, it has a performance cliff.

Interpretation of Cardinality-Slope:

- A **steeper slope** indicates higher sensitivity, meaning that even small cardinality estimation errors can lead to significant deviations in execution costs.
- A **flatter slope** indicates higher robustness, as the plan cost remains relatively stable despite estimation errors.
- The metric is particularly useful for identifying plans that are resilient to errors propagated through deep edges in the query execution tree.

This metric focuses solely on sensitivity, making it ideal for scenarios where the primary concern is mitigating the impact of estimation errors without explicitly balancing cost.

3.1.4.2 The Integral of the PCF - Cardinality Integral

Wolf et al. [154] propose another measure based on the integral of the PCF too (i.e. Cardinality-Integral). The Cardinality-Integral robustness measure is presented as an alternative means to quantify the robustness of query execution plans against cardinality

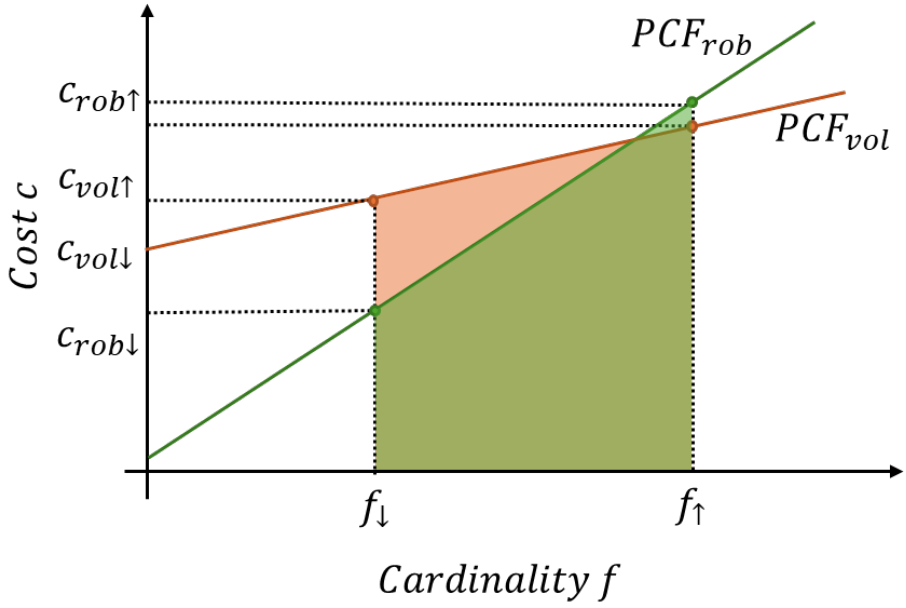


Figure 3.4: The integral of the PCF as a robustness measure

estimation errors. The metric considers both the plan's cost and its sensitivity to changes in cardinality. Unlike metrics based on slope, Cardinality-Integral examines the behavior of the plan's cost over a range of possible cardinalities. It requires a lower and an upper bound for the estimated cardinality and uses the area under the curve of the PCF as the robustness measure for each operator. When evaluated in a range, a plan with a smaller slope might turn out to be more costly than an alternative plan with a steep slope in a wider range of potential cardinality values. An example is provided in Figure 3.4. While the plan represented by PCF_{rob} has a steeper slope than the one represented by PCF_{vol}, its area under the curve in this range is smaller. This means in most points in the range

$[f_{\downarrow}, f_{\uparrow}]$ it has a cheaper cost, therefore, it is more robust.

To compute the Cardinality-Integral for a whole plan, [154] proposes calculating the Cardinality-Integral for each edge in the query plan, within a specified cardinality range. This integral is then multiplied by an edge weighting function that accounts for the sensitivity of different operators to cardinality estimation errors. Finally, the weighted integrals for all edges are summed to produce the Cardinality-Integral value for the entire plan. A lower Cardinality-Integral value signifies a more robust plan.

The incorporation of a cardinality range bounded by a lower and upper limit provides a more realistic assessment of potential cardinality variations. Additionally, the metric's compatibility with any monotonically increasing and differentiable cost function ensures its applicability to diverse cost functions and plan structures.

Interpretation of Cardinality-Integral:

- A **smaller integral value** indicates that the plan exhibits lower sensitivity and cost over the specified cardinality range, reflecting better robustness.
- The metric accounts for realistic estimation errors by considering a bounded range of cardinalities ($[f_{\text{low}}, f_{\text{high}}]$), rather than evaluating robustness at a single point.
- By incorporating cost into the analysis, it provides a more holistic measure of robustness, suitable for scenarios where both stability and efficiency are critical.

This metric is particularly useful when query workloads are expected to experience bounded yet variable cardinality estimates, as it ensures that plans perform well across the entire range.

3.1.4.3 Suboptimality

Robustness in query execution under selectivity estimation errors is also evaluated using measures of suboptimality [51]. These metrics quantify the deviation of the executed plan's performance from the theoretically optimal plan. The three key measures discussed in the literature are *Suboptimality*, *Maximum Suboptimality (MSO)*, and *Average Suboptimality (ASO)*.

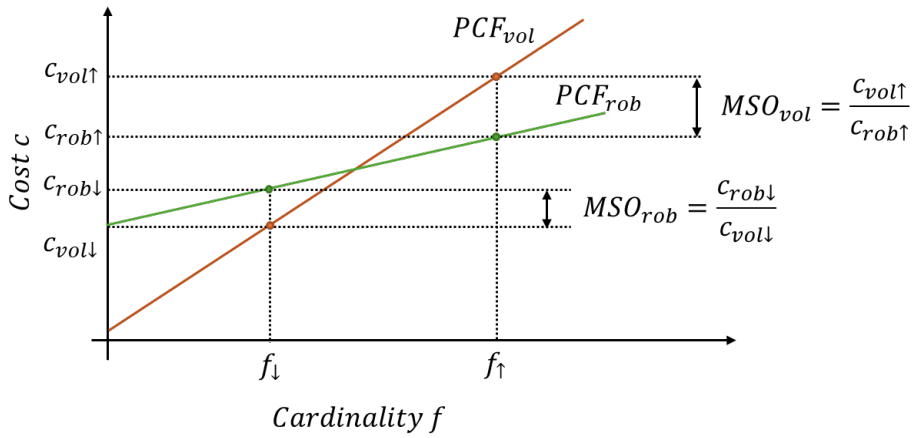


Figure 3.5: Maximum Suboptimality as a measure of plan robustness

Definition 3.4. *The Suboptimality of a plan quantifies the performance degradation when a non-optimal plan is executed at a given query location. Formally, it is defined as:*

$$SubOpt(P, q) = \frac{c_P(q)}{c_{opt}(q)}, \tag{3.1}$$

where:

- P *is the plan being evaluated.*
- q *is the query location (cardinality point).*
- $c_P(q)$ *is the cost of executing plan* P *at location* q*.*
- $c_{opt}(q)$ *is the cost of executing the optimal plan at location* q*.*

The value of SubOpt is always greater than or equal to 1. A value of 1 indicates perfect performance (no degradation), while higher values represent increasing levels of performance loss due to suboptimal plan choices.

Definition 3.5. ***Maximum Suboptimality (MSO)*** *represents the worst-case performance degradation of a plan across all possible cardinality points. It is calculated as the maximum suboptimality over the entire error selectivity space and is formally defined as:*

$$MSO(P) = \max_{q \in ESS} SubOpt(P, q),$$

where:

- P *is the plan being evaluated.*
- ESS *denotes the Error Selectivity Space, encompassing all possible cardinality points.*
- $SubOpt(P, q)$ *is the suboptimality of plan* P *at cardinality point* q*.*

This metric provides a measure of the worst-case robustness of the query execution strategy. Lower MSO values are desirable, as they indicate that the system is resilient even under extreme selectivity estimation errors.

This measure is illustrated in Figure 3.5. Let us assume that the cardinality parameter can take a value in the range $[f_{\downarrow}, f_{\uparrow}]$. At each point, the suboptimality of each plan

compared to the optimal is computed. MSO_{rob} and MSO_{vol} represent the maximum of this ratio for PCF_{rob} and PCF_{vol}, respectively. Since $\text{MSO}_{\text{rob}} < \text{MSO}_{\text{vol}}$, the plan represented by PCF_{rob} is considered more robust in this range.

Definition 3.6. ***Average Suboptimality (ASO)*** *evaluates the expected performance degradation of a plan over the entire error selectivity space. It is defined as:*

$$ASO(P) = \frac{\sum_{q \in ESS} SubOpt(P, q)}{|ESS|},$$

where:

- P *is the plan being evaluated.*
- ESS *denotes the Error Selectivity Space, encompassing all possible cardinality points.*
- $SubOpt(P, q)$ *is the suboptimality of plan* P *at cardinality point* q*.*
- $|ESS|$ *is the size of the error selectivity space.*

This measure captures the average-case performance of the query execution strategy. Lower ASO values indicate that the system performs robustly under typical conditions, complementing the worst-case guarantee provided by MSO.

Interpretation of Suboptimality Metrics: The interplay between *MSO* and *ASO* offers a comprehensive understanding of robustness:

- *MSO* ensures bounded performance degradation in the worst-case scenario, offering guarantees on query execution stability.

- *ASO* provides insights into the system's average-case behavior, ensuring that robustness is not achieved at the expense of everyday performance.

Together, these measures are pivotal in evaluating and comparing the robustness of query execution strategies under uncertainty. For instance, the *plan bouquet approach* achieves significant reductions in both MSO and ASO, demonstrating its effectiveness in robust query processing.

3.1.4.4 Selectivity Error Resistance Factor (SERF)

The *Selectivity Error Resistance Factor (SERF)* is a metric designed to evaluate the robustness of query execution plans in the presence of selectivity estimation errors [51]. SERF quantifies the extent to which a replacement plan improves or worsens the resistance to selectivity errors compared to both the optimizer's original choice and the true optimal plan.

Definition 3.7. *For a query point with an estimated selectivity location q_e and an actual selectivity location q_a, SERF is defined as:*

$$SERF(q_e, q_a) = 1 - \frac{c_{re}(q_a) - c_{oa}(q_a)}{(1+\lambda) \cdot c_{oe}(q_a) - c_{oa}(q_a)},$$

where:

- $c_{re}(q_a)$*: Cost of executing the actual query at* q_a *using the replacement plan.*
- $c_{oa}(q_a)$*: Cost of executing the actual query at* q_a *using the optimal plan.*
- $c_{oe}(q_a)$*: Cost of executing the actual query at* q_a *using the plan chosen by the optimizer at* q_e*.*
- λ*: A user-defined threshold that bounds the allowable cost increase for replacement plans (e.g.,* $\lambda = 0.2$ *allows a 20% increase).*

Interpretation of SERF: The value of SERF provides insight into the effectiveness of the replacement plan in handling selectivity estimation errors:

- **Positive Values** ($0 < \text{SERF} \leq 1$): The replacement plan improves robustness. A SERF value of 1 indicates that the replacement plan performs as well as the true optimal plan.
- **Zero** (SERF = 0): The replacement plan performs equivalently to the optimizer's original choice, with no improvement or degradation.

- **Negative Values** (SERF < 0): The replacement plan degrades robustness, performing worse than the optimizer's original choice.

3.1.4.5 Execution Time Variance (ETV)

A critical measure of robustness in query optimization is the *Variance in Execution Time*, which quantifies the predictability of query performance. Variance captures the degree to which query execution times fluctuate across different runtime scenarios, especially when selectivity estimates are uncertain. Robust query optimizers aim to minimize this variance, ensuring consistent query performance even in the presence of estimation errors.

Definition 3.8. *Let T_i denote the execution time for a query Q in the i-th execution, where n execution times are measured. Each execution time is expected to demonstrate some variance due to changes in the underlying data that lead to more inaccurate cardinality estimates. The variance in execution time, σ_T^2, is defined as:*

$$\sigma_T^2 = \frac{1}{n} \sum_{i=1}^{n} \left(T_i - \mu_T\right)^2,$$

where:

- μ_T *is the mean execution time, given by:*

$$\mu_T = \frac{1}{n} \sum_{i=1}^{n} T_i.$$

- n *is the total number of scenarios or queries considered.*

Lower variance indicates higher predictability, making the system more robust to selectivity estimation errors.

Variance in execution time is an essential metric for evaluating the *consistency* and *predictability* of query plans [6]. While traditional query optimizers often focus solely on minimizing average execution time, users may value predictable performance more than occasional gains in speed. Variance complements average execution time by highlighting the tradeoff between raw performance and stability.

Interpretation of ETV: The work in [6] demonstrates the importance of variance as part of a broader performance-predictability trade-off:

- A **low variance** indicates stable performance, making the system predictable and user-friendly.
- A **high variance** reflects susceptibility to fluctuations caused by selectivity estimation errors, leading to unpredictable query performance.

By incorporating variance into the query optimization process, the authors propose using probabilistic reasoning to balance performance and predictability. Specifically, they introduce a *confidence threshold* parameter that allows users to control this balance. Higher confidence thresholds prioritize plans with lower variance, favoring stability over potential performance gains.

Variance in execution time is a key measure of query robustness, complementing average execution time by addressing predictability. By minimizing variance, robust query optimizers ensure consistent performance, reducing the risk of poor user experiences due to fluctuating query runtimes.

3.1.4.6 Analysis of the Robustness Measures

This section analyzes the robustness measures discussed earlier, highlighting their advantages and limitations in different scenarios.

Cardinality-Slope [154]

Advantages:

- **Evaluates sensitivity to cardinality estimation errors, considering potential propagation of these errors.**

The cardinality-slope metric focuses on the immediate impact of cardinality estimation errors by examining the slopes of the Parametric Cost Function (PCF). This allows it to capture how errors at deeper edges of a query execution plan propagate upward, influencing the costs of higher edges. This propagation insight is valuable for identifying plans vulnerable to cascading errors.

- **Computationally efficient for online optimization scenarios.**

  By relying on the slopes of PCFs rather than integrating or simulating over ranges of cardinalities, this metric can be computed quickly. Its efficiency makes it suitable for online query optimization systems that require real-time decisions without significant overhead.

Limitations:

- **Focuses solely on sensitivity without balancing robustness and cost.**

  The metric's exclusive focus on sensitivity overlooks the actual costs of plans. While a robust plan may have lower sensitivity, it might also incur higher execution costs, leading to a suboptimal balance between performance and predictability.

- **May overemphasize errors on deeper plan edges.**

  Since the metric naturally highlights deeper edges in the plan tree due to their error propagation potential, it may disproportionately penalize plans with complex

structures, even if these plans perform well under typical conditions.

Cardinality-Integral [154]

Advantages:

- **Balances robustness and cost by integrating over a range of cardinalities.** Unlike the cardinality-slope metric, the cardinality-integral metric accounts for both robustness and cost by evaluating the cumulative impact of cardinality estimation errors across a bounded range. This provides a more comprehensive assessment of plan performance under realistic scenarios.

- **Suitable for realistic scenarios where cardinality ranges are bounded.** By integrating over a specified range $(f_{\text{low}}, f_{\text{high}})$, the metric reflects practical conditions where estimation errors fall within predictable bounds. This makes it particularly relevant for workloads with bounded uncertainty.

Limitations:

- **Computationally more intensive due to the need for integration.** Calculating integrals over cardinality ranges increases the computational overhead compared to slope-based metrics, which may limit its applicability for real-time query optimization scenarios.

- **Depends on accurate estimates of cardinality bounds ($f_{\mathbf{low}}, f_{\mathbf{high}}$).**

  The metric's accuracy heavily relies on the reliability of the specified cardinality range. Inaccurate bounds can lead to misleading robustness assessments, undermining its utility.

Maximum Suboptimality [31]

Advantages:

- **Provides a worst-case performance guarantee by identifying the maximum degradation due to selectivity estimation errors.**

  This metric ensures that the chosen plan performs within a predictable bound under the most adverse estimation errors, making it ideal for critical applications where worst-case performance is a priority.

- **Useful for scenarios where ensuring a bounded worst-case execution time is critical.**

  In time-sensitive systems, such as real-time analytics or high-stakes decision-making applications, the worst-case guarantee provided by MSO enhances reliability and user confidence.

Limitations:

- **Focuses only on the worst-case, potentially overlooking average-case robustness.**

  By emphasizing the worst-case scenario, the metric may lead to overly conservative decisions, ignoring plans that perform well in most cases but poorly under extreme conditions.

- **Can be overly pessimistic, especially for well-distributed query workloads.**

  In scenarios where cardinality errors are evenly distributed, the worst-case focus can unnecessarily penalize plans that are otherwise robust under typical conditions.

Average Suboptimality [31]

Advantages:

- **Represents the typical performance degradation, making it suitable for average-case optimization.**

  This metric provides a balanced view of robustness by focusing on the overall performance across the error space, making it ideal for scenarios with less variability in workload patterns.

- **Captures the overall robustness of a query execution plan.**

  By averaging the performance degradation across all possible scenarios, it offers a comprehensive measure of robustness that aligns with real-world expectations.

Limitations:

- **Does not provide guarantees for worst-case scenarios.**

  While the metric excels in average-case evaluations, it lacks the ability to ensure

performance under extreme estimation errors, which may be unacceptable in critical applications.

- **Sensitive to outliers in query performance.**

  Extreme outliers can disproportionately influence the average, leading to skewed robustness assessments and potentially suboptimal plan selection.

Selectivity Error Resistance Factor [51]

Advantages:

- **Quantifies how well a replacement plan performs compared to the true optimal plan.**

  SERF directly evaluates the improvement or degradation of a replacement plan relative to the optimal plan, offering actionable insights into its robustness.

- **Balances robustness and cost by considering a user-defined confidence threshold (λ).**

  The confidence threshold provides flexibility, allowing users to specify their tolerance for cost increases in exchange for improved robustness, tailoring the metric to different requirements.

Limitations:

- **Requires detailed cost information for the replacement, optimal, and original plans.**

  Accurate computation of SERF depends on comprehensive cost estimates, which may not always be readily available or reliable in practice.

- **Interpretation can be less intuitive for end users compared to simpler metrics.**

  The mathematical formulation and reliance on cost ratios make SERF harder to understand, limiting its accessibility for non-technical stakeholders.

Execution Time Variance [6]

Advantages:

- **Directly quantifies predictability, which is a critical aspect of robustness in practical systems.**

  Variance in execution time highlights the consistency of plan performance, providing a straightforward measure of predictability that is crucial for user satisfaction.

- **Simple to compute and interpret.**

  Unlike other metrics that require complex calculations or simulations, execution time

variance is easy to compute and interpret, making it suitable for quick assessments.

Limitations:

- **Does not capture the magnitude of performance degradation, focusing only on consistency.**

  While predictability is important, the metric does not address how far a plan deviates from optimal performance, limiting its utility for comprehensive robustness evaluations.

- **Requires sufficient execution data to compute variance reliably.**

  Accurate variance computation depends on a substantial dataset of execution times, which may not be available for new or rarely executed queries.

Table 3.2 summarizes the advantages and limitations of each robustness measure.

3.1.4.7 The Impact of Plan Structure on Robustness

The structure of the physical execution plan impacts its robustness in different ways. A well-studied aspect is related to the cardinality estimate error propagation patterns. Ioannidis and Christodoulakis [61] explore the problem of error propagation in the size estimation of join results. The authors present a formal framework for analyzing how errors

Measure	Advantages	Limitations
MSO [31]	Provides worst-case guarantees. Suitable for critical applications.	Overly pessimistic. Ignores average-case performance.
ASO [31]	Represents typical performance degradation. Captures overall robustness.	Lacks worst-case guarantees. Sensitive to outliers.
SERF [51]	Balances robustness and cost. Quantifies replacement plan performance.	Requires detailed cost information. Interpretation can be complex.
ETV [6]	Simple and intuitive. Measures predictability directly.	Ignores magnitude of performance degradation. Requires execution data.
CS [154]	Evaluates sensitivity to cardinality errors. Computationally efficient.	Focuses only on sensitivity. May overemphasize deeper plan edges.
CI [154]	Balances robustness and cost. Suitable for bounded cardinality ranges.	Computationally intensive. Requires accurate cardinality bounds.

Table 3.2: Comparison of Robustness Measures

in database statistics affect query result estimates, showing that errors often propagate exponentially with the number of joins. This observation has important implications for the robustness of the query plans. For example, as demonstrated by Wolf et al. [154], the error in a cardinality estimate of a join operator close to the bottom of a deep plan propagates all the way toward the root of the plan. This leads to several misestimated operator costs. These misestimations accumulate and lead to a larger cardinality and cost errors for the whole plan. This in turn implies that inaccurate estimations are more consequential to robustness when the target operator is closer to the bottom of a deep plan.

3.1.5 Robust Statistics

Cardinality estimation is a cornerstone of query optimization in modern database management systems (DBMS). Accurate cardinality estimates are critical for generating efficient query execution plans, as they guide the query optimizer in selecting appropriate join orders, access paths, and physical operators. However, achieving precise estimates is a challenging endeavor due to inherent limitations in available statistics at compile-time.

Cardinality estimation methods often rely on simplifying assumptions to approximate complex data distributions [23]. These assumptions include:

- **Uniformity Assumption**: Data values within a predicate range are uniformly distributed.
- **Independence Assumption**: Attributes involved in a query are statistically independent.
- **Inclusion Assumption**: Values in one attribute or join key are fully contained within another.

While these assumptions enable computationally tractable models, they fail to capture the intricacies of real-world data, such as skewness, correlations, and outliers. Conse-

quently, the accuracy of cardinality estimates deteriorates when these assumptions are violated, leading to suboptimal query execution plans [23].

Moreover, data distributions are not static. Over time, they are subject to shifts or drifts due to changes in workloads, user behaviors, or evolving datasets. Traditional cardinality estimation methods, which depend heavily on static statistics, struggle to adapt to such changes, further exacerbating estimation errors [58].

To address these challenges, robust cardinality estimation has emerged as a promising approach. Robust methods aim to:

1. **Minimize Reliance on Simplifying Assumptions**: Using advanced statistical models or machine learning techniques, robust estimators model data distributions more accurately.

2. **Reduce Sensitivity to Data Drift**: They employ adaptive mechanisms to dynamically update and recalibrate models in response to changes in the underlying data.

3. **Reduce Sensitivity to Workload Shift**: They aim to reduce sensitivity to changing workload patterns by improving generalizability.

4. **Quantify Estimation Uncertainties**: Robust methods often provide confidence

intervals or probabilistic bounds, enabling query optimizers to make informed decisions despite imperfect knowledge.

Robust cardinality estimation represents a shift towards more adaptable, accurate, and reliable query optimization strategies, particularly in environments characterized by dynamic data and complex query workloads.

Studies concerning robust cardinality estimation can be broadly categorized into two groups based on whether they employ ML in their proposed approaches. The first group includes studies that do not use ML, in this thesis refereed to as *Traditional* approaches. The second group includes studies that employ some form of ML. The following sections provide an overview of these two groups of studies.

3.1.5.1 Traditional Approaches to Robust Cardinality Estimation

Before the rise of machine learning-based solutions, traditional approaches leveraged probabilistic and statistical methods to improve cardinality estimation. These methods aimed to mitigate inaccuracies arising from naive assumptions like uniformity and independence, which often led to suboptimal query plans.

Types of Statistics Traditional methods for cardinality estimation can be categorized based on the type of statistical information they employ.

One approach focuses on **univariate probability distributions**, addressing the uniformity assumption. These methods typically use histograms to model attribute distributions more accurately, with variants such as equi-depth and frequency histograms providing refined granularity. Poosala et al. proposed an extensive taxonomy of histograms and introduced improved histograms for selectivity estimation of range predicates, incorporating techniques like V-Optimal and Compressed Histograms to reduce approximation errors [123]. Chaudhuri et al. presented an adaptive page sampling algorithm for constructing equi-height histograms [18]. Aboulnaga and Chaudhuri developed self-tuning histograms that dynamically refine themselves based on query feedback, making them particularly effective in large databases [3]. Ioannidis provided a historical overview of histogram techniques, highlighting their evolution in practical implementations [62]. Bruno and Chaudhuri demonstrated the utility of histograms on intermediate query results for refining selectivity estimates [13].

In contrast, **multivariate probability distributions** tackle the independence assumption by modeling attribute dependencies. Techniques such as multidimensional histograms and pairwise column group statistics capture joint selectivities for correlated attributes. Poosala and Ioannidis introduced multi-dimensional histograms to capture joint data distributions using space-partitioning heuristics [124]. Lee et al. proposed compressed histograms to efficiently model multidimensional distributions [86]. Deshpande

et al. introduced dependency-based histograms that capture correlations using variance-based splitting criteria [28]. Ilyas et al. developed the CORDS framework for automatic discovery of correlations and soft functional dependencies [60].

Types of Discovery Mechanisms Another way to categorize traditional methods is based on their mechanisms for discovering statistical information.

Workload-driven approaches build statistics based on observed intermediate query expressions, adapting to recurring query patterns. Bruno et al. proposed STHoles, a multidimensional histogram structure that dynamically adjusts to workload patterns and captures high-density regions efficiently [14]. Chaudhuri et al. introduced random sampling techniques for histogram construction to balance efficiency and accuracy [18]. Bruno and Chaudhuri later extended this idea with Statistics on Intermediate Tables (SITs), which leverage intermediate query results [13].

In contrast, **data-driven approaches** construct joint probabilities directly from the data by identifying correlations and functional dependencies. Ilyas et al.'s CORDS framework used chi-squared analysis to detect statistical relationships and soft dependencies [60]. Giannella et al. introduced mining frequent patterns from data streams using tilted-time window frameworks, enabling efficient discovery of correlations in dynamic data [41]. Hertzschuch et al. developed sampling techniques that address highly selective queries by

incorporating uncertainty models [54].

Probabilistic Cardinality Estimation Seppi et al. pioneered the use of Bayesian reasoning in database query optimization, developing a probabilistic framework for handling uncertainty in cost and selectivity estimates [132]. Poosala et al. explored singular value decomposition (SVD) for approximating joint data distributions, offering an alternative to traditional histograms [124]. Haas et al. introduced sampling-based estimators for the number of distinct attribute values [48]. Chaudhuri et al. proposed sampling algorithms optimized for constructing equi-height histograms under practical constraints [18]. Building on these advancements, Babcock and Chaudhuri developed a robust cardinality estimation framework using beta distributions for selectivity estimates [6].

Despite their contributions, probabilistic approaches share some limitations with other traditional methods. Scalability remains a significant challenge, especially for high-dimensional data. Adaptability to dynamic workloads or unobserved query patterns is limited. Moreover, incorporating user preferences in the trade-off between predictability and performance often requires explicit parameter tuning, such as adjusting the confidence threshold.

3.1.5.2 Learning-based Approaches to Robust Cardinality Estimation

As discussed above, the traditional approaches to cardinality estimation rely on histograms [124], sampling [6], or other statistical methods to approximate cardinalities [55]. Such approaches often face limitations when dealing with complex queries, skewed data, or evolving workloads. Learning-based cardinality estimation has emerged as a promising approach, leveraging machine learning techniques to overcome these challenges by capturing intricate data correlations and query patterns.

Learning-based approaches can be broadly categorized into **data-driven** and **query-driven** methodologies. While these studies share a common goal of enhancing the accuracy and robustness of cardinality estimation, they differ fundamentally in their learning paradigms and data dependencies:

- **Data-driven approaches** focus on directly modeling the underlying data distribution in an unsupervised manner. These methods aim to approximate the joint probability distribution of attributes within a database, often employing advanced deep learning architectures such as autoregressive models [163], Gaussian Mixture Models [105], or normalizing flows [148]. By learning from the data itself, these methods excel in handling unseen queries and adapting to dynamic data distributions, making them robust against workload shifts.

- **Query-driven approaches**, in contrast, utilize supervised learning techniques by training on sample queries paired with their ground-truth cardinalities. These methods emphasize understanding query patterns and workloads, often incorporating advanced neural networks, attention mechanisms, or reinforcement learning to model complex query behaviors [79, 93]. While they require labeled query data, query-driven methods are particularly effective in optimizing query execution plans for specific workloads, achieving high accuracy in environments with well-defined query patterns [127].

This section delves into advances in learning-based cardinality estimation, exploring the unique contributions of both data-driven and query-driven studies. By highlighting their strengths, limitations, and applications, we aim to provide a comprehensive overview of how these methodologies are reshaping the landscape of query optimization in modern database systems.

Data-Driven Learned Cardinalities Data-driven cardinality estimation techniques have emerged as a promising alternative to traditional methods. These techniques leverage machine learning (ML) to learn the data distribution and provide more accurate cardinality estimates. This section explores the different types of data-driven models that have been proposed in the literature.

Kernel Density Estimation (KDE) KDE is a non-parametric method that estimates the probability density function of a dataset by averaging Gaussian kernels centered around sample points. KDE-based models have been employed to estimate multidimensional selectivity by averaging local probability distributions. Early work by Blohsfeld et al. [11] which demonstrated KDE's potential for single-dimensional range query selectivity estimation, especially with sufficiently smooth data. Gunopulos et al. [47] extended KDE to multi-dimensional selectivity estimation, showing comparable accuracy to histograms. Heimel et al. [53] developed a self-tuning, GPU-accelerated KDE estimator that utilizes numerical optimization for bandwidth selection and adapts to database changes, achieving better performance than traditional KDE and multi-dimensional histograms. KDE faces some challenges: high inference latency due to scanning all sample points, difficulty handling discrete and string data, and the complexity of estimating join selectivities efficiently. The application of GPU-accelerated KDE estimators, even though they demonstrate improved inference times, remains practically limited since the use of GPUs in database systems is not yet prevalent.

Deep Autoregressive Models Deep autoregressive models are a type of neural network that can learn complex data distributions. Naru is a single-table cardinality estimator that leverages deep autoregressive models to capture correlations among all columns with-

out making the independence assumption. It accomplishes this by approximating the joint data distribution in its full form, eliminating the need to specify combinations of columns for synopses [164]. NeuroCard extends Naru to handle multi-table join queries [163]. It learns the correlations across multiple joins in a single deep autoregressive model. NeuroCard is particularly effective for handling queries with a tree-structured join schema. However, deep autoregressive models can be computationally expensive to train and may require a large amount of storage space. This approach has been shown to achieve significant improvements in accuracy over other methods. NeuroCard is particularly effective for handling queries with a tree-structured join schema. However, deep autoregressive models can be computationally expensive to train and may require a large amount of storage space [148].

Sum-Product Networks (SPNs) SPNs are a type of probabilistic graphical model that can represent high-dimensional probability distributions compactly. DeepDB uses a specific type of SPN called a Relational Sum-Product Network (RSPN) for cardinality estimation [58]. SPNs can capture complex dependencies between attributes, but they can be challenging to learn and may not generalize well to unseen queries.

Factorized Sum-Product Networks (FSPNs) FSPNs are an extension of SPNs that aim to improve speed and accuracy by combining different types of factorization

methods. FLAT is a cardinality estimation method built upon FSPNs, designed for fast probability computation, lightweight model size, and accurate estimation quality [180]. It can handle both single-table and multi-table join queries.

Normalising Flow Normalising Flow models learn a transformation from a simple distribution to a more complex one, allowing them to model intricate data distributions effectively. FACE, a cardinality estimator leveraging normalizing flow, focuses on learning a continuous joint distribution for relational data [148]. FACE can estimate the cardinality for sequential and parallel queries, addressing the need for high inference efficiency in data-driven methods. However, as with many deep learning models, optimizing the model architecture and hyperparameters for specific datasets and query workloads can be challenging.

Gaussian Mixture Models (GMMs) and Autoregressive Models Integrated Autoregressive Model (IAM) combines GMMs and autoregressive models for cardinality estimation. This approach leverages GMMs to reduce the domain size of continuous attributes, making the search space for the autoregressive model smaller and improving efficiency [105]. However, as with any mixture model approach, the number of components and the initialization of parameters can impact the model's performance.

Query-driven Learned Cardinality This section reviews key machine learning architectures employed in query-driven cardinality estimation, discussing their strengths, limitations, and thematic contributions to the field. The subsections correspond to distinct architectural approaches, highlighting their applicability and performance across diverse scenarios.

Multi-Layer Perceptrons (MLPs) / Fully Connected Neural Networks (FCNNs) Early research explored using Multi-Layer Perceptrons (MLPs), also known as Fully Connected Neural Networks (FCNNs), for selectivity estimation. Lakshmi et al. [84] investigated neural networks for selectivity estimation in extensible databases, focusing on training neural networks to learn the cumulative distribution functions (CDFs) of attribute values. This method facilitated selectivity estimation for various predicates. Lu and Setiono [98] expanded on this approach, demonstrating that three-layer neural networks could effectively learn these CDFs, enabling accurate query size estimations for both selection and join operations.

In 2015, Liu et al. [95] proposed a neural network architecture that used bounded ranges on each column as input, successfully learning a function to estimate selectivities for queries with relational predicates. Their method showed significant improvement over traditional cardinality estimators, particularly in handling correlated and skewed data.

However, directly applying MLPs or FCNNs to selectivity estimation presents limitations, especially for complex queries involving multiple predicates. Recent research highlights that while these models can capture basic relationships, they struggle with complex interactions in high-dimensional data spaces. Dutt et al. [33] suggested that although neural networks can divide the query space into local regions for accurate estimation, the number of required regions grows exponentially. This growth increases model complexity, training time, and memory requirements. Additionally, these models are heavily dependent on the training workload, limiting their ability to generalize to unseen query patterns or data distributions.

Multi-Set Convolutional Networks (MSCN) Kipf et al. first introduced the Multi-Set Convolutional Network (MSCN) for cardinality estimation in their 2019 work [79]. MSCN is designed to effectively represent relational query plans using set semantics, enabling the model to capture query features and true cardinalities without sensitivity to the order of joins. This design leads to smaller models and improved predictions. Their experiments on real-world datasets demonstrated that MSCN significantly enhances the quality of cardinality estimations, a critical factor for query optimization.

Building on this foundation, Wu and Cong [157] presented UAE, a unified deep autoregressive model that incorporates MSCN to leverage both data and query workloads for

learning the joint data distribution. UAE employs MSCN to capture query features and enhance estimation accuracy, particularly for complex queries with multiple joins.

Kipf et al. further explored the application of MSCN in their 2019 study [80], where they introduced Deep Sketches as compact database representations. They utilized MSCN as the core model for estimating cardinalities by incorporating information about qualifying base table samples alongside static query features.

Recognizing the importance of query featurization for ML-based cardinality estimation, Müller et al. [111] proposed novel query featurization techniques and evaluated their impact on various ML models, including MSCN. Their work highlighted how different featurization techniques influence MSCN's ability to capture predicate information, ultimately improving estimation accuracy.

However, the increasing adoption of ML-based cardinality estimation models has also raised security concerns. Zhang et al. [171] investigated vulnerabilities of query-driven estimators like MSCN to poisoning attacks. Their proposed PACE attack effectively manipulated training data, inducing significant errors in MSCN's estimations.

Addressing limitations in handling data and workload drift, Negi et al. [115] introduced Robust-MSCN. They incorporated modifications to MSCN's query representation and training techniques, enhancing its robustness against workload and data drifts. Their

model demonstrated improved resilience and consistently outperformed PostgreSQL in query performance, even under dynamic workload conditions.

Recurrent Neural Networks (RNN) Query-driven cardinality estimation, a crucial aspect of query optimization, has witnessed innovative approaches leveraging the power of Recurrent Neural Networks (RNNs). Early exploration in this domain is evident in the work by [83], which introduced DREAM, a deep RNN model designed to handle the complexities of approximate substring queries. DREAM utilizes a packed learning method for efficient training and demonstrates promising accuracy in estimating cardinalities for this specific type of query.

Moving beyond specialized query types, researchers explored the use of RNNs for more general cardinality estimation tasks. Chen et al. [22] proposed E2E, a hybrid model employing RNNs to process SQL queries as sequences. By encoding queries in a left-deep tree structure, E2E captures dependencies across different query components, enabling a comprehensive understanding of query semantics for enhanced cardinality estimation.

Further advancements in RNN-based approaches led to models estimating both cost and cardinality simultaneously. Sun and Li [138] presented a Tree-structured model, which, while not explicitly identified as a Tree-LSTM, exhibits architectural and functional similarities to this RNN variant. This model effectively captures the hierarchical relationships

within query plans, enabling accurate estimation of both cost and cardinality.

The pursuit of efficiency alongside accuracy is evident in the work of [147], which introduced LPCE-I, an RNN-based model resembling Tree-LSTM. Designed to process execution plans recursively, this approach not only improves the accuracy of cardinality estimates but also enhances the speed of the estimation process.

Overall, these studies highlight the growing interest in leveraging RNNs and their variants for query-driven cardinality estimation. RNNs, with their ability to capture sequential and hierarchical relationships, offer a promising avenue for tackling the complexities inherent in query plans and data distributions. However, the computational cost associated with training and inference, especially for large and complex datasets, remains a potential limitation of RNN-based approaches.

Tree-Based Ensemble Methods Researchers have explored the use of tree-based ensemble methods, like Gradient Boosting Decision Trees (GBDT) and XGBoost, for query-driven cardinality estimation. The paper "Selectivity Estimation for Range Predicates using Lightweight Models" [33] was an early adopter of this approach. The authors found that lightweight models based on GBDT can offer fast and accurate estimates for multi-dimensional range predicates, surpassing the performance of traditional methods used in database systems at the time. However, the study primarily focused on range predicates

and acknowledged that further research was needed to extend the applicability of these models to other types of queries.

Building on this foundation, the paper "Automating Localized Learning for Cardinality Estimation Based on XGBoost" [177] delved deeper into leveraging XGBoost for enhanced accuracy. The authors proposed a localized learning approach where, instead of building a single global model, multiple specialized local models are automatically constructed based on the characteristics of the query workload. This automation addressed the challenge of manually specifying query patterns for local models, improving the practicality and usability of the approach. The study demonstrated that this automated localized learning with XGBoost could lead to significant improvements in estimation accuracy compared to existing learning alternatives.

Further advancing the practicality of learned cardinality estimation, AutoCE [170] proposed a novel system called AutoCE, which acts as a model advisor to guide the selection of the most suitable cardinality estimation model for a given dataset. Recognizing that no single model consistently outperforms others across all datasets and data distributions, AutoCE aimed to simplify the process of choosing the best model for a specific situation. This approach, while not directly focused on tree-based ensembles, highlights the increasing maturity of the field and the shift towards developing systems that can effectively leverage various learned models, including those based on tree-based ensemble methods,

for practical cardinality estimation tasks.

Tree-based ensemble methods, particularly XGBoost, offer several advantages for query-driven cardinality estimation. Their inherent ability to capture non-linear relationships and handle high-dimensional data makes them well-suited for modeling the complex interactions between query features and cardinalities. In addition, their efficiency in training and inference contributes to the practicality of these methods in real-world database systems. However, the potential for overfitting, especially when dealing with limited or biased training data, remains a consideration. Furthermore, unlike deep learning models, tree-based ensemble models grow in size as the complexity of the training data increases. This in turn may lead to long inference times. Therefore, careful tuning and appropriate regularization techniques are essential to mitigate these risks and ensure the robustness, generalizability, and efficiency of tree-based ensemble models for cardinality estimation.

Attention Mechanisms A rising trend in query-driven cardinality estimation is the incorporation of attention mechanisms, inspired by their success in natural language processing and computer vision. These mechanisms allow models to focus on the most relevant parts of the input query and data, enhancing their ability to capture complex relationships and improve estimation accuracy.

[150] utilize attention mechanisms within their cardinality estimator module. They

argue that traditional methods struggle with data skewness, leading to inaccurate estimations. By incorporating attention, their model weighs different features of the input query differently, effectively mitigating the impact of skewed data and achieving better generalization. However, the effectiveness of attention mechanisms can be sensitive to the choice of hyperparameters and the specific architecture used.

Building upon this concept, [93] take a step further by utilizing attention mechanisms to address the challenge of dynamic workloads, where data and query patterns change over time. Their approach incorporates a data-encoder module with self-attention to learn complex interactions between different attributes in the database, allowing the model to adapt to evolving data distributions and maintain accurate estimations. While attention mechanisms improve the model's adaptability, they also introduce additional complexity to the architecture, potentially increasing training and inference time.

Graph Neural Networks (GNN) Graph Neural Networks (GNNs) have emerged as a promising approach for query-driven cardinality estimation, particularly in scenarios involving complex join queries. [128] exemplify this trend by introducing the JGMP (Join Graph Message Passing) model. This model leverages the ability of GNNs to represent queries as join graphs, effectively capturing the intricate relationships between tables and predicates. By performing message passing over the join graph structure, JGMP learn

complex interactions and dependencies relevant for accurate cardinality estimation. The authors emphasize the sample efficiency of their approach, demonstrating that JGMP can achieve significant improvements in estimation accuracy even when trained on limited data, a crucial advantage in practical settings where acquiring extensive training data can be challenging.

While GNNs hold significant potential, certain limitations merit consideration. The computational complexity associated with GNN training and inference, especially for large and highly interconnected graphs, can pose challenges in real-world database systems. Furthermore, the effectiveness of GNNs depends on the quality and representativeness of the join graph structure. Inaccurate or incomplete graph representations can adversely affect estimation accuracy. Despite these challenges, the application of GNNs for cardinality estimation, as showcased by [128], represents a significant step toward leveraging the power of graph-based learning for enhancing query optimization in database systems.

Probabilistic Deep Learning A prominent theme in query-driven cardinality estimation is the exploration of probabilistic deep learning techniques, driven by the need for models that can quantify and manage uncertainties inherent in predicting query result sizes. These approaches not only aim for accurate cardinality estimations but also provide insights into the confidence levels of their predictions, leading to more informed decisions

in query optimization.

[157] introduced a novel method leveraging Neural Network Gaussian Processes (NNGP). By integrating Bayesian principles into the deep learning architecture, NNGP inherently quantifies uncertainty in its predictions. This uncertainty awareness allows for a more robust approach to cardinality estimation, as it flags potential low-confidence estimations and provides insights for further model refinement or adaptation.

Further solidifying the focus on uncertainty management, [97] presented a system utilizing deep ensembles. Built on an ensemble of deep neural networks, this method not only predicts the cardinality but also estimates the associated uncertainty by aggregating predictions from individual models. This ensemble approach effectively captures various sources of uncertainty, leading to more reliable estimations and a better understanding of potential risks associated with specific predictions.

[121] took a different probabilistic approach by employing Bayesian Networks (BN). Their work highlighted the strength of BNs in representing and reasoning with uncertainty within a structured graphical model. This framework facilitates the explicit modeling of dependencies between data attributes and provides a principled way to propagate uncertainty through the network, resulting in more robust cardinality estimations.

Finally, [157] proposed a hybrid approach combining a Masked Autoencoder (MAE)

with Gaussian Mixture Models (GMM). This fusion leverages MAE's capacity to learn data distributions and GMM's flexibility in handling complex densities, resulting in a more robust and uncertainty-aware cardinality estimator. The probabilistic nature of GMMs further enhances the ability to quantify and manage uncertainty, leading to improved reliability in its predictions.

In summary, the integration of probabilistic principles into deep learning models for cardinality estimation marks a significant step toward developing more robust and reliable systems. The ability to quantify and manage uncertainty through methods like NNGP, deep ensembles, Bayesian Networks, and hybrid approaches offers valuable insights into the confidence of cardinality estimations, paving the way for more informed and resilient query optimization strategies.

Query-driven vs. Data-driven Cardinality Estimation Query-driven and data-driven approaches represent two prominent paradigms in learned cardinality estimation, each with its own strengths and limitations.

Query-driven approaches Query-driven approaches leverage supervised learning techniques to train models that map query characteristics to their estimated cardinalities. This approach relies heavily on the availability of a training query workload with associated

true cardinalities [87, 98, 127].

- **Advantages of query-driven approaches:**
  - **Efficiency in inference:** These models typically offer fast inference times, making them well-suited for online query optimization scenarios [115].
  - **Support for diverse query types:** Query-driven methods can effectively handle various filter types, including string operations, disjunctions, and complex join patterns [115].
- **Disadvantages of query-driven approaches:**
  - **Dependence on training workload:** The performance of these models is heavily reliant on the quality, diversity, and representativeness of the training queries [50, 115, 163, 180]. Query-driven methods are prone to the "well-known workload shift issue," where they struggle to generalise to query workloads with different distributions than the training set [50].
  - **Vulnerability to workload shifts:** Changes in query patterns or data distributions can significantly impact the accuracy of query-driven models [87, 115]. Addressing workload shift often necessitates retraining or adapting the model, which can be resource-intensive [115].

Data-driven approaches Data-driven approaches, in contrast, aim to learn the underlying data distribution of the database without relying on specific query workloads [87, 128, 148]. They treat each tuple as a sample from a joint probability distribution and build models to capture these distributions.

- **Advantages of data-driven approaches:**
  - **Generalisation to unseen queries:** By learning the data distribution, data-driven models can effectively estimate the cardinality of queries that were not present in the training workload [180].
  - **Robustness to workload shifts:** These models are generally less susceptible to changes in query patterns as they are not tied to a specific workload [148]. Data-driven methods can outperform query-driven approaches, especially when dealing with more complex datasets and query workloads [50].
- **Disadvantages of data-driven approaches:**
  - **Scalability to large joins:** Accurately modeling the joint distribution for large joins can be computationally expensive and lead to large model sizes [87].
  - **Inference latency:** While data-driven models generalize well, the inference process may be slower compared to query-driven approaches [115]. This is

particularly relevant for OLTP workloads, where fast cardinality estimation is crucial [50]. However, inference latency has a less significant impact on OLAP workloads, where long-running queries dominate execution time [50]. Therefore, data-driven models might be more suitable for OLAP scenarios.

- **Support for Complex Queries:** Many existing methods have limited support for complex queries, such as those involving string-matching predicates, disjunctive predicates, or nested queries [163].
- **Sensitive to data drifts:** Data in real-world databases is constantly changing. While some methods support updates, efficiently adapting models to data changes and evolving query workloads remains an active research area [50].

The evolution of learned cardinality estimation research shows distinct trends in distribution and publication frequency of both data-driven and query-driven approaches, as illustrated in Figure 3.6. Over the past several years, there has been significant growth in this research area, with the number of publications steadily increasing year over year, demonstrating the growing recognition of machine learning's potential to address long-standing challenges in cardinality estimation. When examining methodological approaches, we observe that query-driven techniques initially dominated the field, likely due to their more straightforward implementation and alignment with traditional supervised learning

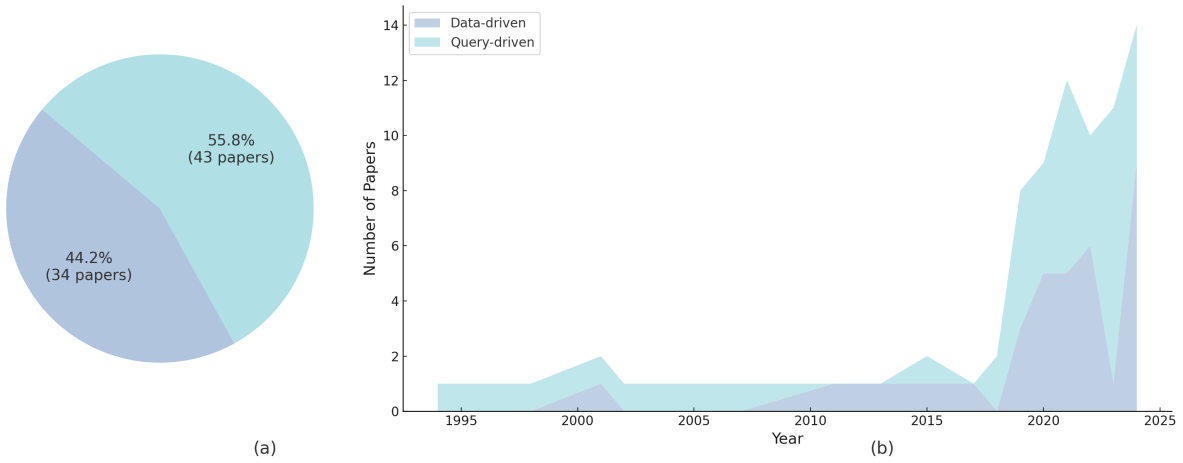


Figure 3.6: ML-based cardinality estimation studies (a) distribution across Data-driven and Query-driven and (b) number of studies per year

paradigms. However, data-driven approaches have gained considerable momentum in recent years, as researchers recognize their superior generalization capabilities and robustness to workload shifts. This trend reflects a deeper understanding within the database community that modeling the underlying data distribution — rather than just query patterns — may provide more sustainable solutions for the cardinality estimation problem, particularly in environments with dynamic workloads and evolving data characteristics.

In summary, query-driven methods are often preferred for their efficiency and ability to handle diverse query types, but they are susceptible to workload shifts. Although data-driven methods offer better generalization and robustness to workload shifts, they can face challenges with robustness to data drifts, scalability, and inference latency, especially for large joins. The choice between the two approaches depends on the specific requirements

of the application and the characteristics of the data and the query workload.

3.1.6 Robust Cost Models

A cost model translates plan-specific features (e.g., cardinalities, operator types, or join graphs) into a numeric estimate of query execution cost. Even small errors in these estimates or mismatches in the model's assumptions can cause severe performance cliffs. Efforts on robust cost modeling aim to mitigate sensitivity to inaccuracies by combining iterative feedback loops, sophisticated statistics, explicit risk/uncertainty modeling, and often pretraining from broader sources.

3.1.6.1 Adaptive Feedback-Driven Cost Correction

These methods iteratively revise cost models by monitoring actual runtime behavior and folding that information into subsequent cost estimates. In early work [19], a system adjusts selectivity predictions after each query execution, preventing repeated misestimations for future queries. LEO [137] captures cardinality discrepancies at runtime in DB2, refreshing internal statistics so incorrect estimates are not reused. Recent approaches often steer or refine a legacy optimizer using partial or repeated plan runs. Solutions such as [113, 172] gather plan outcomes (sometimes mid-execution) to correct cost parameters, averting drastically wrong plans on future invocations. For shifting data or workloads,

Warper [89] partially retrains local submodels whenever drift is detected, while Negi et al. [115] draws on-the-fly cardinalities from sub-queries to keep cost estimates realistic. Hint-based frameworks like HERO [181] attempt different plan hints in sequence, discarding any that cause notable regressions, and revert to safer defaults upon detecting large deviations. Additionally, Neo [101] initializes its learned cost model from a traditional optimizer, continuously refining its parameters through each execution outcome. Likewise, Bao [100] leverages repeated latency observations to update a neural performance predictor, adopting Thompson-sampling-like plan selection and trimming expansions if the true runtime diverges excessively. Another approach Eraser [152] introduces a plug-in that flags suspicious cost estimates using a multi-stage "risk check" and excludes them if prior queries show large errors. Such feedback-driven corrections integrate relatively seamlessly with existing optimizers and steadily improve cost accuracy when queries recur or overlap.

3.1.6.2 Statistical and Ensemble-Based Estimators

These methods improve cost-model robustness by capturing richer statistics or combining multiple sub-predictors. Rather than relying on simpler or uniform assumptions, they aim to maintain accuracy across correlated or skewed data scenarios and diverse workloads. Early efforts [90] employ robust regression to accommodate outliers in training data, making cost predictions less fragile when selectivity estimates deviate. Tzoumas et al. [144] use

adaptive graphical models to refine selectivity dependencies among predicates, mitigating the impact of sudden distribution changes. Later, Yang et al. [164] propose a flow-based unsupervised model to capture joint attribute distributions, preventing abrupt errors for unusual filter combinations. Meanwhile, Neo [101] employs row-wise embeddings based on column values (akin to word2vec [106]) to handle unseen predicates. DeepDB [58] and NeuroCard [163] emerged as deep generative cardinality estimators, capturing multi-table correlations more accurately. Similarly, Kaoudi et al. [73] attempt cross-engine cost regression, bridging partial cost differences across DBMSs. This cross-system perspective helps contain misestimates on engines or workloads lacking specialized local statistics. Bayesian generative insights appear in BayesCard [49], consolidating data distribution into a single estimator to avoid abrupt plan shifts when data deviate. Fauce [97] uses multiple sub-predictors to combining deep ensembles to hedge single-model outliers. Extended spatio-textual distribution tracking emerges in LATEST [121], which captures location- or text-driven correlations that would elude standard cost formulas, thus preventing abrupt performance drops for queries with unusual textual or spatial filters. ALECE [93] adopts an attention-based aggregator for SPJ queries on dynamic workloads, focusing on capturing evolving predicates or partial correlations across tables via multi-head self-attention. Meng et al. [105] fuses Gaussian mixtures with autoregressive modeling to handle mixed distributions more flexibly, thus strengthening cost predictions when partial aspects of

the data deviate from typical assumptions. These advanced or ensemble-based estimators significantly reduce average errors, effectively capturing correlations and varied data patterns.

3.1.6.3 Risk- and Uncertainty-Aware Cost Modeling

A direct strategy for robustness is to predict confidence intervals or risk distributions for each plan cost, then leverage these uncertainties to avoid or penalize high-variance outcomes. An early approach SkinnerDB [143] integrates regret-bounded reinforcement learning, continuously measuring partial execution returns and halting catastrophic plans mid-run. Its multicore extension SkinnerMT [151] generalizes the same regret-based pruning to parallel environments, thus systematically curbing the worst-case overhead from misestimations.

Several approaches [21, 68, 97, 175] adopt approximate Bayesian or probabilistic neural networks (e.g., Monte Carlo Dropout) to obtain an uncertainty estimate along with cost predictions. By repeatedly sampling network weights at inference, the variance of predicted cost signals the cost model's uncertainty while inferring. Plans flagged with high predicted variance can either be verified (e.g., by partial execution or sampling) or demoted to traditional methods if runtime resources are limited. Meanwhile, Yu et al. [166] use multi-head attention [145], treating each attention head's output as a sub-predictor, and aggregates

them to quantify spread in cost estimates. This variance is then used to pick safer fallback plans when the cost difference among heads is large. Other works [17, 68, 97, 166, 173, 175] incorporate neural network Gaussian processes (NNGP) [37]. These models output a predictive distribution over plan cost; the distribution's variance reveals the model's confidence to the prediction. Large variances trigger fail-safe logic: either the plan's cost is inflated by a "risk penalty" to steer the optimizer toward more conservative alternatives, or the system reverts to a known, robust plan from a legacy cost model. In both cases, uncertain predictions are contained, preventing performance cliffs.

Additionally, Thirumuruganathan et al. [141] propose cost intervals at plan granularity, removing plan edges where intervals become extremely wide. Ensemble-based posteriors or dropout-based distributions also appear in LEON [21], which refines cost predictions at each enumeration step by recomputing posterior distributions over candidate plan expansions. Under dynamic conditions, Negi et al. [115] blend partial data sampling with broader predictive bounds to cap the worst-case plan cost. Another method, Kepler [30], leverages spectral normalized neural Gaussian processes (SNGP) [96] to detect out-of-distribution queries: if the confidence radius is exceeded, it immediately discards or backtracks to a default plan. Similarly, LOGER [20] integrates explicit "no large regressions" constraints into a learned optimizer, ensuring that uncertain transformations are avoided if they could cause significant run time blow-ups. In addition, some works [35, 160] track plan-level uncertainty

intervals, penalizing edges associated with large uncertainty. Meanwhile, Eraser [152] applies a two-stage check that first labels uncertain or contradictory cost estimates using approximate checks, then removes them from the final search space. These risk-focused methods more reliably avert extreme plan regressions but do incur overhead in estimating distributions, implementing fallback, and managing uncertain queries.

3.1.6.4 Pretrained or Zero-Shot Learned Models

These techniques address the issue of cold-start or domain mismatch by exploiting large-scale pretraining or cross-domain knowledge to establish a more robust baseline. Rather than training a cost model from scratch for each new database or query workload, they begin with generalizable representations that can be adapted with little or no retraining, mitigating catastrophic errors when local data are sparse or rapidly changing. An early cross-platform study [73] aligns partial cost regressors across different DB engines, effectively transferring knowledge of operator cost behaviors. By consolidating cost patterns from multiple systems, their approach avoids large mispredictions when one engine's native cost model is inadequate. A broader concept of universal modeling appears in [159], which trains a single cost predictor to adapt under different hardware or data distributions. The model retains a "core" representation across tasks and fine-tunes only minor layers for each new environment, significantly reducing the risk of initial plan regressions.

Zero-shot estimation is explored in [57], combining multiple historical sources to produce immediate out-of-the-box predictions without any local examples. This strategy prevents severe early misestimates by leveraging prior signals from analogous workloads. When domain conditions differ, minimal local tuning refines cost estimates. Similarly, domain adaptation in [150] ports a learned cardinality model to new data by adjusting only domain-specific layers, preserving the main structure from the original training. Meanwhile, Yang et al. [161] repurpose classical cost logic as a prior instead of building learned models from scratch, thus avoiding radical errors on queries similar to the baseline's scope.

In terms of system context shifts, Negi et al. [116] proposes a pretrained Transformer that can incorporate OS- or hardware-level signals in order to remain robust under changing system conditions. Others [35,169] combine pretrained embeddings with local calibration or uncertainty intervals, thus providing safe fallback boundaries for queries in new databases. By retaining general patterns from large offline corpora, these pretrained or zero-shot approaches reduce catastrophic cost predictions at the start of deployment, substantially narrowing the initial "blind spot."

3.1.6.5 Merits of Robust Cost Models

Improved Handling of Complex Data Distributions: By leveraging advanced statistics, ensembles, or generative models, many robust cost estimators handle correlated,

skewed, or rapidly evolving data more gracefully. Compared to naive histograms or uniform assumptions, they can capture subtle distributions or cross-attribute dependencies, thus lowering the probability of large misestimation events.

Reduced Performance Cliffs: Techniques that incorporate explicit risk or uncertainty estimates (e.g., Bayesian neural networks, neural network Gaussian processes, or dropout-based variance) help an optimizer avoid high-variance plans. This mechanism prevents query performance from abruptly deteriorating due to outlier conditions, leading to more consistent execution times.

Dynamic Adaptation to Drifts or Feedback: Feedback-driven or hint-steering approaches continuously incorporate runtime observations. Over repeated queries or partial subplans, they converge to better estimates, diminishing the effects of stale or inaccurate statistics. This adaptability is particularly valuable when data distribution or workload characteristics shift over time.

Early-Stage Safety for New Workloads: Pretrained or zero-shot solutions start from a robust generic baseline, mitigating catastrophic errors at initial deployment. They significantly reduce blind spots for newly arrived queries or novel systems, giving the database more stable performance even with minimal local retraining.

Complementary to Legacy Optimizers: In many instances, robust cost models

function as an overlay to existing query optimizers, adding uncertainty quantification or adaptive feedback loops without discarding the optimizer's original heuristics. This complementary design provides both forward-looking estimates and safe fallback behaviors.

Potential for Enhanced User Trust: By explicitly modeling uncertainties and offering fallback or risk penalties, these approaches can boost confidence in cost predictions. Users may more readily rely on an optimizer that demonstrates self-awareness regarding out-of-distribution queries and uncertain plan estimates, thereby fostering overall trust in the system.

3.1.6.6 Limitations of Robust Cost Models

Complexity: Many of these approaches require significant architectural changes or extensive additional components (e.g., specialized data structures, ensemble sub-models, or plan hint frameworks). For instance, maintaining large Bayesian neural networks or generative autoregressive estimators demands nontrivial engineering overhead and complex pipeline integration. Similarly, advanced domain adaptation or pretraining methods may involve rewriting large segments of the optimizer or training infrastructure.

Overhead: Techniques that rely on frequent feedback loops, risk-based gating, or sampling/distribution modeling often add runtime overhead. For example, maintaining approximate posteriors or performing partial plan execution to verify uncertain cost estimates

can noticeably slow optimization in short-running queries. Storing plan-level variance, plus fallback logic, also increases both memory footprint and computational cost.

Dependence on Representative Data: Ensemble-based or generative strategies inherently rely on acquiring enough relevant training data to cover typical query patterns. When data distributions drift or new query templates appear, these learned cost models can degrade, unless specialized retraining or domain-adaptation techniques are deployed. The same limitation applies to pretraining-based methods that assume cross-domain similarities.

Limited Efficacy for Non-Repetitive Workloads: Feedback-driven solutions and partial-run verification gain robust accuracy over repeated queries or subplans. If a workload is highly diverse or ephemeral, they may not accumulate enough feedback to correct inaccurate cost estimates, leaving them vulnerable to performance cliffs.

Fallback and Risk Tuning Challenges: Many robust solutions propose fallback to legacy cost models or inflation of predicted cost when uncertainty is high. Setting thresholds or risk-penalty magnitudes is tricky: an aggressive fallback may lead to frequent usage of suboptimal default plans, while an overly lenient one risks catastrophic regressions. Determining these balances often involves trial-and-error or ad-hoc heuristics.

Scalability to Complex Queries: Methods like Bayesian neural networks or en-

semble generative modeling scale better than naive histograms, but extremely large query graphs (e.g., many tables or advanced predicates) can still outstrip available training resources. In practice, building robust cost estimates for queries with dozens of joins, multiple correlated attributes, or user-defined predicates remains challenging without further hierarchical decompositions or domain-specific constraints.

Maintaining Robustness under System Shifts: Finally, changes to hardware, concurrency levels, or resource allocation can invalidate cost assumptions. Risk- or uncertainty-based gating can mitigate some unpredictability, but many solutions still assume stable hardware or concurrency conditions unless explicitly retrained or reconfigured. In dynamic environments (e.g., cloud auto-scaling, mixed transactional-analytical loads), ensuring consistent cost-model robustness continues to be nontrivial.

3.1.7 Robust Search Algorithms

A robust search algorithm is a method designed to identify query execution plans that perform consistently well across diverse and uncertain conditions. It typically selects the most reliable option from a set of candidate plans by evaluating their estimated costs and potential variability. These algorithms aim to minimize performance degradation caused by inaccuracies in assumptions, such as cardinality estimates or selectivity predictions, during query optimization. Balancing exploration (evaluating diverse plans) and exploitation

(choosing the most promising plan) is a fundamental challenge in their design.

3.1.7.1 Dynamic Programming

Dynamic Programming (DP) [131] is a foundational method in query optimization for enumerating candidate plans. DP systematically evaluates all subsets of relations to determine the most cost-effective join order for each subset. It begins by constructing optimal plans for smaller subsets and incrementally combines them to generate plans for larger subsets. By storing intermediate results for subsets in a lookup table, DP avoids redundant computations, significantly improving efficiency. As it exhaustively explores all possible join orders, DP theoretically guarantees an optimal plan, provided that the cardinality estimates and cost model are sufficiently accurate. Moreover, exploration and exploitation are balanced in DP since it explores all possible join orders and exploits the best result in the lookup table.

3.1.7.2 Top-K Search

The top-k search algorithm is a type of query processing used in databases to efficiently find the top k results that best satisfy a given query, based on a scoring or ranking criterion. Instead of retrieving all possible results and then sorting them, the algorithm directly focuses on retrieving only the top k results, optimizing performance and reducing compu-

tational overhead. Balsa [162] uses a count-based exploration to safely evaluate new plans during training. This approach prioritizes executing previously unseen plans from a pool of likely promising candidate plans generated by beam search. The beam search returns top k plans sorted by their predicted latency. Balsa executes the best plan among these k that has not yet been executed, promoting exploration within a "trust region" of high-quality plans. Once all k plans have been executed, Balsa shifts to exploitation by selecting the plan with the lowest predicted latency. This method ensures a balance between exploring new options and leveraging the most effective known plans.

Similar to Balsa, LEON [21] also aims to find top-k candidate plans. LEON formalizes query optimization as a pairwise ranking problem, focusing on correctly ordering candidate plans rather than predicting exact execution costs. This reduces the impact of errors in cost estimation. Furthermore, to improve exploration efficiency, LEON incorporates an uncertainty-aware mechanism. Plans with higher uncertainty in their rankings are prioritized for execution, ensuring that the optimizer learns effectively from areas of the search space where it is less confident. LEON exemplifies a robust search algorithm in query optimization by balancing exploration and exploitation, maintaining consistent performance under uncertainty.

3.1.7.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) effectively balances exploring less-visited branches and focusing on high-performing paths, ensuring comprehensive coverage of the search space without excessive computation. HyperQO [166] adapts MCTS as a search strategy to optimize query plan selection. It ensures robustness by leveraging a hybrid approach that combines cost-based and learning-based methods to generate diverse candidate plans. By integrating these two approaches, HyperQO minimizes the risk of selecting suboptimal plans caused by over-reliance on a single methodology. Additionally, the framework incorporates uncertainty-aware plan selection, enhancing robustness against errors in learning-based predictions and improving performance for out-of-distribution queries.

3.1.7.4 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) formulates query optimization as a sequential decision-making problem, where an agent learns to select efficient query execution plans through interaction with a database system. DRL improves its decision-making process and avoids inefficient join orders that could lead to high-cost execution plans. A key application of DRL in query optimization is Join Order Selection, where the model learns from execution feedback, making query optimization more adaptive and resilient to estimation errors.

However, DRL requires input representation that accurately reflects the environment to generate high-quality actions. This often necessitates additional networks for state representation. Several approaches have been proposed to enhance state encoding for join trees, DQ [81] and Rejoin [102] use Multi-Layer Perceptrons to represent the state, but they struggle to differentiates semantically similar trees, as they assign identical representation to different join trees. To address this limitation, RTOS [168] leverages Tree-LSTM [140] to capture a more representative state. Despite its advantages, DRL-based query optimization demands significant computational resources, as it relies on extensive trial-and-error learning, making practical deployment a challenge.

3.1.7.5 Bio-Inspired Algorithms

Bio-inspired algorithms draw inspiration from natural processes and behaviors to solve optimization problems. In the context of query optimization, several bio-inspired approaches have been explored, including genetic algorithms, ant colony optimization, and swarm intelligence methods. Genetic algorithms (GAs) simulate the process of natural selection to evolve optimal solutions over generations [9], maintaining a population of candidate query execution plans and iteratively refining them through selection, crossover, and mutation operations [82, 117]. Ant Colony Optimization (ACO) algorithms are inspired by the foraging behavior of ants, where artificial ants traverse a graph representing the query

optimization problem, depositing pheromones on paths that lead to good solutions [34,42], and have been applied to multi-join query optimization [63,92] and distributed query processing [142,176]. Swarm intelligence approaches, including Particle Swarm Optimization (PSO) and Artificial Bee Colony (ABC) algorithms, model the collective behavior of social insects and animals [8,74], representing potential solutions as particles or bees that adjust their positions based on individual and group experiences to find optimal query plans [4,5].

While these bio-inspired approaches offer innovative perspectives on plan search and optimization, they present significant limitations that make them less suitable for robust query optimization in practice. The algorithms typically require substantial computational resources due to their iterative nature and population-based approaches, suffer from premature convergence to local optima in complex search spaces, and are highly sensitive to parameter settings that are difficult to tune appropriately for different query scenarios and database characteristics.

3.1.7.6 Merits of Robust Search Algorithm

Dynamic Programming Merits Dynamic Programming (DP) provides several key advantages in robust query optimization. DP ensures a globally optimal solution, provided that the cost model is accurate, making it theoretically sound for query optimization problems. The algorithm efficiently handles large search spaces by incrementally processing

subsets and reusing solutions from smaller problems to build larger ones, avoiding redundant computations through its lookup table mechanism. DP systematically evaluates all subsets of relations to determine the most cost-effective join order, providing comprehensive coverage of the search space while maintaining computational efficiency through memoization.

Top-K Search Merits Top-K search algorithms offer several benefits for robust query optimization. They efficiently retrieve the top-K solutions, ensuring high-quality results within the top subset while focusing computational resources on the most promising candidates. These algorithms concentrate on evaluating the top-ranked candidates, eliminating less promising options early in the optimization process. Top-K approaches provide a good balance between exploration and exploitation, allowing the optimizer to consider multiple high-quality alternatives rather than committing to a single plan, which enhances robustness against estimation errors.

Monte Carlo Tree Search Merits Monte Carlo Tree Search (MCTS) provides several advantages for robust query optimization. MCTS constructs a search tree, selectively exploring branches with the potential for optimality while avoiding exhaustive enumeration of all possible plans. The algorithm efficiently handles large search spaces by selectively expanding promising branches of the search tree, prioritizing computational resources on

high-potential regions. MCTS effectively balances exploring less-visited branches and focusing on high-performing paths, ensuring comprehensive coverage of the search space without excessive computation, making it particularly suitable for complex query optimization scenarios.

Deep Reinforcement Learning Merits Deep Reinforcement Learning (DRL) approaches offer several benefits for robust query optimization. DRL formulates query optimization as a sequential decision-making problem, allowing the system to learn from execution feedback and adapt to changing query patterns and data distributions. The approach improves decision-making over time, avoiding inefficient join orders that could lead to high-cost execution plans. DRL methods can handle complex state representations and learn sophisticated policies that may not be easily captured by traditional rule-based or cost-model approaches, potentially providing more adaptive and resilient optimization strategies.

Bio-Inspired Algorithm Merits Bio-inspired algorithms, while having significant limitations for robust query optimization, do offer some potential benefits in certain contexts. These algorithms can explore complex search spaces through their population-based or swarm-based approaches, potentially discovering novel optimization strategies that might not be found through traditional methods. Genetic algorithms can maintain diversity in

their candidate solutions, reducing the risk of getting trapped in local optima in some optimization landscapes. Ant colony optimization can effectively model the query optimization problem as a graph traversal problem, potentially finding good solutions through pheromone-based path selection mechanisms.

3.1.7.7 Limitations of Robust Search Algorithm

Dynamic Programming Limitations Dynamic Programming (DP) presents several limitations despite its theoretical optimality guarantees. DP requires maintaining a lookup table for storing intermediate results, which demands significant memory for large search spaces. The exponential growth in the number of possible join orders with increasing table count makes DP computationally prohibitive for queries involving many tables. Additionally, DP depends heavily on accurate cost models and cardinality estimates, where errors can propagate through the optimization process and lead to suboptimal results. The method's exhaustive exploration approach, while ensuring optimality, becomes impractical for complex queries due to its computational overhead.

Top-K Search Limitations Top-K search algorithms rely critically on the accuracy and quality of their ranking functions. Poor rankings can result in suboptimal plan selections, particularly when the ranking criteria do not accurately reflect actual execution perfor-

mance. The effectiveness of top-K algorithms is limited by the quality of the underlying cost model used for ranking, making them vulnerable to the same cardinality estimation errors that affect other optimization approaches. Additionally, the fixed value of K may not be optimal for all query types, potentially excluding better plans or including inferior ones depending on the specific query characteristics.

Monte Carlo Tree Search Limitations Monte Carlo Tree Search (MCTS) faces several challenges in query optimization contexts. When the search tree has high branching factors (many possible actions at each step), MCTS leads to higher computational costs due to the need to simulate multiple possible outcomes to evaluate nodes. The algorithm's effectiveness depends on the quality of the simulation function used to evaluate leaf nodes, and inaccurate simulations can lead to poor plan selections. Additionally, MCTS requires careful tuning of its exploration-exploitation balance, and the algorithm's performance can be sensitive to parameter settings such as the number of simulations per node.

Deep Reinforcement Learning Limitations Deep Reinforcement Learning (DRL) approaches to query optimization present several significant limitations. DRL demands substantial computational resources due to its reliance on extensive trial-and-error learning, making practical deployment challenging. The approach requires input representation that accurately reflects the environment to generate high-quality actions, often necessitating

additional networks for state representation. DRL methods struggle with semantically similar trees, as they may assign identical representations to different join trees, limiting their ability to distinguish between similar but distinct query plans. The learning process can be slow and requires substantial training data, making it less suitable for dynamic environments where query patterns change frequently.

Bio-Inspired Algorithm Limitations Bio-inspired algorithms present several fundamental limitations that make them unsuitable for robust query optimization. They typically require significant computational resources due to their iterative nature and the need to evaluate numerous candidate solutions, making them impractical for real-time query optimization scenarios. These algorithms often suffer from premature convergence to local optima, especially in complex or high-dimensional search spaces, which is particularly problematic in query optimization where the search space can be vast and highly irregular. The performance of bio-inspired algorithms is highly dependent on parameter settings (e.g., mutation rates, pheromone evaporation rates, swarm sizes), and tuning these parameters appropriately for different query scenarios and database characteristics is challenging and often requires extensive experimentation. As the complexity and size of databases grow, the efficiency of bio-inspired algorithms degrades significantly, with exponential growth in search space complexity making these methods computationally prohibitive for large-scale

or distributed database systems. Unlike traditional optimization methods, bio-inspired algorithms do not provide deterministic guarantees about solution quality or convergence time, which is problematic for robust query optimization where consistent performance is essential. Finally, incorporating bio-inspired algorithms into existing query optimizers requires significant architectural changes and may conflict with established optimization frameworks and cost models.

3.1.8 Robust Execution

Robust execution in the context of database systems means ensuring predictable query performance even in the face of errors or changing conditions. A key aspect is robust query processing, focusing on mitigating the impact of selectivity estimation errors, which often lead to suboptimal execution plans. The following subsections review studies of robustness execution from three different perspectives, adaptive plan execution, discovery-based approach, and re-optimization techniques.

3.1.8.1 Adaptive Plan Execution

Adaptive plan execution generally refers to systems that can adjust the execution strategy of a query during its runtime, responding to changing conditions or observed data characteristics. The concept of adaptive query processing is changing the system resources

during query execution when encountering the effects of unreliable cardinality estimation. It typically involves modifying or adapting the execution plan during runtime based on observed conditions. This technique usually depends on the type of the query and the underlying DBMS. **Current research focuses on different purposes of adaptivity.** For instance, Sharkova et al. [133] focus on stream processing systems, which needs an adaptive mechanisms to handle dynamic, real-time data streams. The paper suggests an adaptive optimization framework that predicts upcoming data statistics using trends observed from previous streaming windows, enabling better cost-based optimization. Kader et al. [65] focuses on XQueries, they use a join graph to represent queries, which encapsulates XML navigation, relational joins and predicates, and the optimizer executes operators in the join graph sequentially, materializing partial results and using adaptive chain sampling to explore multiple execution paths ahead. While exploring, it adapts dynamically to runtime data properties to minimize intermediate result sizes. Other work [126] and [64] focus on RDBMS. Raman et al. [126] presents a novel algorithm aimed at improving the efficiency and robustness of RID-list (row ID list) intersection operation, particularly in star joins, column stores, and search engines. It proposes lazy merging, RID-lists are not fully constructed upfront. Instead, segments of the lists are formed and merged on demand during intersection. It then introduces adaptive intersection where use dynamic, n-ary intersection strategy and adjusts the intersection process based on runtime RID density observations,

thereby mitigating the impact of misestimated cardinalities. Jintao et al. [64] introduce adaptive data pruning by dynamically restructuring data ranges and aligning them iteratively to maximize pruning benefits. By reducing unnecessary data, the system minimizes errors in cardinality estimation, preventing error propagation during execution.

3.1.8.2 Discovery-based

Discovery-based typically refers to an adaptive strategy that incrementally discovers and adjusts the query execution plan by partitioning the problem space geometrically, often leveraging runtime feedback. The discovery-based approach uses information observed during query execution to iteratively refine the execution process, ensuring better alignment with the actual data properties and runtime conditions. Cardinality misestimation has long been a significant challenge in query optimization, often resulting in drastically suboptimal performance. To address this, Dutt et al. [31] proposed the PlanBouquet approach, which provides provable worst-case performance guarantees. Instead of relying on selectivity estimates, PlanBouquet employs a runtime discovery mechanism that uses a "bouquet" of plans to identify the appropriate plan based on the actual selectivities encountered during execution. It constructs multi-dimensional space, covering entire selectivity ranges between these predicates. It then generates a set of plans, called the "bouquet", chosen from the optimal plans within the created multi-dimensional space. This bouquet ensures

that at least one plan is near-optimal for any selectivity combination. At runtime, the actual selectivities are discovered through a series of partial executions of the bouquet plans. It executes the plan with the cheapest cost, and it switches to the next plan when the cost budget exceeded, or it finishes within the budget. As the algorithm progresses through the IC, it moves closer to the optimal plan for the given selectivity and ensures performance guarantees. With the success of PlanBouquet, Karthik et al. [77] propose SpillBound, which retains the core contour-wise discovery approach of PlanBouquet, but introduces significant improvements. It enhances PlanBouquet's multi-dimensional space but provides a more powerful half-space-based pruning technique. This refinement enables more efficient handling of the search space, leading to significant improvements in query optimization performance.

Instead of generating a set of plans among various ranges of selectivity, Xiu et al. [160] propose a penalty-aware model PARQO, which uses a stochastic optimization approach to minimize the penalty incurred by a plan compared to the true optimal plan, considering uncertainty in selectivity estimates. PARQO utilizes workload-informed profiling to build statistical error models, and analyze dependencies among query predicates, which allows PARQO to perform sensitivity analysis and identify sensitive selectivity dimensions that have the biggest impact on performance, which could provide users with insights.

3.1.8.3 Re-optimization

Query re-optimization is a dynamic approach in database query processing where a query plan, initially generated at compile time, is adjusted during execution based on actual runtime conditions. Babu et al. [7] proposes RIO, which uses bounding boxes to represent statistical uncertainty, identifies robust and switchable plans to minimize runtime re-optimization and incorporates runtime sampling to refine the estimates dynamically, which makes the statistics accurate and reliable during processing. Another work, POP [125], which uses checkpoint operators (CHECK) to compare estimated and actual cardinalities, and the execution pauses and reoptimization might occur if significant discrepancies are detected. While re-optimization can improve query execution by adapting to runtime conditions, most research has focused on minimizing the need for re-optimization due to its associated costs (e.g., computational overhead, interruption of query pipelines)

3.1.8.4 Merits of Robust Execution

Robust execution is designed to ensure that a query runs efficiently and predictably even in the presence of conditions that can cause unexpected performance degradation. We analyze from different perspectives:

Improved Robustness: Techniques like Plan Bouquets [31] and SpillBound [77] aim

to enhance query processing robustness by addressing the challenge of inaccurate selectivity estimation. These methods deploy runtime selectivity discovery, half-space pruning, and leveraging the concavity of plan cost functions to provide performance guarantees even in the presence of estimation errors.

Reduce Optimization Overhead: PARQO [160] strive to reduce the computational cost associated with robust query optimization. PARQO aims to minimize expected penalty by leveraging statistical models of selectivity estimation errors, leading to more focused and efficient optimization compared to approaches considering the entire selectivity space.

Adaptivity to Changing Conditions: Methods like [133] and [129] enable query processing systems to adapt to evolving data characteristics and system environments. Both works could leverage prediction based on past query executions, or historical performance data to re-optimize queries for subsequent runs, enabling adaptation to different hardware.

3.1.8.5 Limitations of Robust Execution

While these works demonstrate significant achievements, they also have certain limitations:

Computation Overhead: Some robust query optimization methods can be computa-

tionally expensive when the queries have high dimensionality in the error-prone selectivity space. This is particularly true for techniques that require extensive exploration of the selectivity space, such as PlanBouquets [31] and SpillBound [77]. Work on reoptimization have its own challenges, each re-optimization invocation requires the system to re-invoke the query optimizer, and if re-optimization occurs frequently, the overhead of multiple optimizer calls can significantly impact the overall query execution time.

Implementation Complexity: Implementing adaptive and robust techniques often involves modifications to the core database engine, which can be complex and intrusive. Especially on work like [77] and [65] that need run-time resource control for optimization.

3.2 Approximate Probabilistic Machine Learning

Machine Learning is a field that studies methods that learn from experience to improve performance on some set of tasks [107]. It is a subfield of Artificial Intelligence that studies intelligence demonstrated by machines rather than natural beings [27]. The main objective of a machine learning model is to learn from a set of samples called the 'training set' and generalize to unseen samples called a 'test set'. Deep learning is a subfield of machine learning and studies models based on artificial neural networks. Its development was motivated by tremendous generalization capabilities when dealing with high-dimensional

data in problems such as image or speech recognition [43].

Basic machine learning and deep learning models usually do not provide uncertainty estimates for their predictions. This means the basic models cannot tell how certain they are about the predictions they make. Over the past few years, there has been a surge of interest in the deep learning research community to understand and quantify the predictive uncertainty rooted in different sources [2, 40]. Gawlikowski et al. [40] identify various sources of uncertainty as follows:

1. **Variability in a real-world situation:** The natural variability in the environment that affects parameters such as temperature, illumination, clutter, etc., which in turn can change the expression of objects.

2. **Error and noise in the measurement system:** The precision of the measurement systems is variable. Therefore, the measured values, either in features of the object representation or the labels, may carry various levels of error.

3. **Errors in the model structure:** The structure of the model, for example, the number of parameters that determine the model's memorization capacity, can lead to under- or over-fitting on the training data, hence resulting in an under- or over-confident model.

4. **Errors in the training procedure:** The training process is usually stochastic

and depends on various parameters such as batch size, learning rate, regularization, etc. Therefore, two models with identical architectures are very unlikely to produce identical predictions.

5. **Errors caused by unknown data:** The training data is usually limited in its coverage of the whole space of potential inputs. When the model is provided with a sample drawn from a different distribution than the training data's distribution, the prediction will be uncertain. An extreme example of an out-of-distribution case is when a model is trained to classify images of cats and dogs and is provided with an image of a bird for prediction.

Predictive uncertainty is usually classified into two types: *data* (or *aleatoric*) uncertainty and *model* (or *epistemic*) uncertainty [59]. The variability of the behavior of the real-world phenomena under investigation is explained by data uncertainty. In machine learning, data uncertainty can also be caused by noisy input vectors or labels, due to limitations in the measurement system, as well as low-dimensional input vectors that do not suffice to explain the sample properly. These factors are considered irreducible, as even the best models cannot eliminate or reduce them. Model uncertainty, however, captures the variability of predictions, which is caused by a lack of knowledge about the best model. This is rooted in limitations in model architecture, the training procedure, or the coverage

of the training examples [40]. These factors are in principle reducible, for example, by better tuning of parameters or by improving the coverage of the training data.

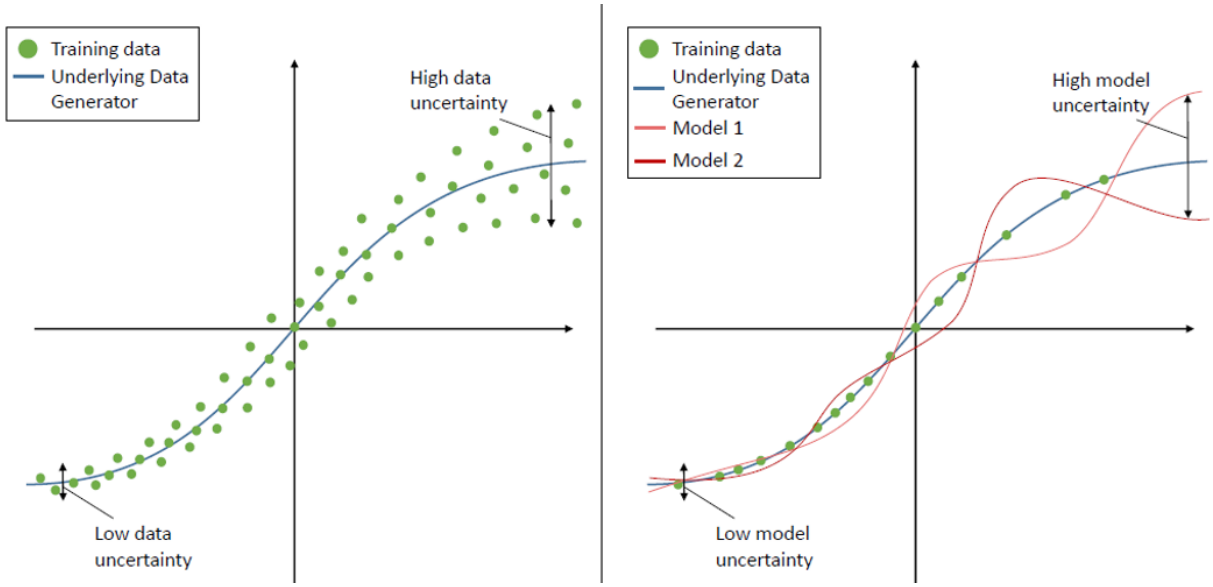


Figure 3.7: Data and model uncertainty in regression models [40]. The x-axis represents the input feature, and the y-axis represents the predicted values

Figure 3.7 illustrates data and model uncertainties in regression tasks. In these plots, the x-axis represents the input feature, and the y-axis represents the predicted values. Note from the left diagram that even the best model, which fits the underlying data generator, will have uncertain predictions (i.e., data uncertainty). Also note from the right diagram that in ranges of x with a lower density of points, the disagreement between alternative models (i.e., model uncertainty) increases. Due to similar observations, Hüllermeier and Waegeman [59] point out that increasing the dimensionality of input data such that the

variability of the target is better explained reduces data uncertainty, while it can increase model uncertainty as it adds to the complexity of the model to be learned. In addition, they point out that increasing the number of samples in areas with low coverage improves model uncertainty.

3.2.1 Uncertainty Modeling

Bayesian inference is a powerful framework for understanding predictive uncertainties. In this framework, the model uncertainty is formalized as a probability distribution over the model parameters θ given the training data D. The data uncertainty is formalized as a probability distribution over the model outputs y^* given a parameterized model f_θ and an input sample x^*. The predictive distribution can then be obtained by inference through marginalization [40]:

$$p(y^* \mid x^*, D) = \int \underbrace{p(y^* \mid x^*, \theta)}_{\text{Data}} \underbrace{p(\theta \mid D)}_{\text{Model}} d\theta \tag{3.2}$$

In other words, the second term $p(\theta \mid D)$ in equation (3.2) describes the variability inherent to the model, while the first term $p(y^* \mid x^*, \theta)$ describes the variability inherent to the outputs regardless of the model parameters. While $p(\theta \mid D)$ cannot be computed analytically, it is approximated by variational parameters $q(\theta)$. This forms the basis for

Bayesian neural networks (BNNs). BNN-based approaches are proposed as an extension of standard neural networks to model predictive uncertainties. They model the network parameters as variables with joint probability distributions (usually Gaussian) and aim at inferring the parameters of this distribution (mean and covariance matrix in the case of Gaussian distribution).

This can be done in principle by minimizing the Kullback-Leibler (KL) divergence, which quantifies the level of dissimilarity between the two distributions $q(\theta)$ and $p(\theta \mid D)$ as follows:

$$\mathrm{KL}(q(\theta) \parallel p(\theta \mid D)) = E_{\theta \sim q(\theta)}\left[\log(q(\theta))\right] - E_{\theta \sim q(\theta)}\left[\log(p(\theta \mid D))\right] \tag{3.3}$$

where all expectations are taken with respect to $q(\theta)$. The approximation $q^*(\theta)$ obtained in this way is the setting from the variational family $q(\theta)$ which is closest to the posterior $p(\theta \mid D)$. However, the computation of the above KL divergence is intractable since the second term $\log(p(\theta \mid D))$ requires computing $\log(p(D))$. This can be made explicit by expanding equation (3.3):

$$\mathrm{KL}(q(\theta) \parallel p(\theta \mid D)) = E_{\theta \sim q(\theta)}\left[\log(q(\theta))\right] - E_{\theta \sim q(\theta)}\left[\log(p(\theta, D))\right] + \log(p(D)) \tag{3.4}$$

Since $\log(p(D))$ is constant with respect to $q(\theta)$, instead, a proxy function is optimized,

which is called the evidence lower bound (ELBO):

$$\text{ELBO}(q) = E_{\theta \sim q(\theta)} \left[\log(p(\theta, D)) \right] - E_{\theta \sim q(\theta)} \left[\log(q(\theta)) \right] \tag{3.5}$$

Note that maximizing the ELBO is equivalent to minimizing the KL divergence. The idea of inferring $p(\theta \mid D)$ from a family of variational distributions $q(\theta)$ by the above optimization is called Variational Inference (VI) [10]. In deep learning, loss functions such as cross-entropy for classification tasks and mean squared error for regression tasks are inspired by or related to maximizing the log-likelihood for an assumed distribution [40].

In recent years, various techniques have been developed in the deep learning field for capturing data and model uncertainties, as extensively reviewed by several studies [2,40,59]. In the following sections, the methods used in this research for quantifying the predictive uncertainties will be discussed.

3.2.2 Aleatoric (Data) Uncertainty

As was shown by [119], with a Gaussian prior, a neural network can be designed in such a way that it predicts the parameters of the normal distribution. In other words, it not only predicts the conditional expected value but also the conditional variance of the target given the training data and input sample. This is done by adding a second branch to the

output of the neural network that predicts the variance. Then, optimizing the following loss function corresponds to maximizing the log-likelihood with a Gaussian prior, as shown in [119]:

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\log\left(\hat{\sigma}_i^2\right) + \frac{(y_i - \hat{\mu}_i)^2}{2\hat{\sigma}_i^2} + \frac{1}{2}\log 2\pi\right) \tag{3.6}$$

where $\hat{\mu}_i$ is plugged in from the first branch of the output layer, $\hat{\sigma}_i^2$ is plugged in from the second branch, and y_i is plugged in from the labels. This approach is in line with the Variational Inference framework.

Note that maximizing the log-likelihood can be viewed as minimizing the dissimilarity between the empirical distribution defined by the training set and the model distribution, with the degree of dissimilarity measured by the Kullback-Leibler (KL) divergence [43]:

$$\text{KL}(p(\theta \mid D) \parallel q(\theta)) = E_{\theta\sim p(\theta|D)}\left[\log(p(\theta \mid D))\right] - E_{\theta\sim p(\theta|D)}\left[\log(q(\theta))\right] \tag{3.7}$$

where $p(\theta \mid D)$ represents the distribution of the underlying data-generating process, and $q(\theta)$ represents the approximated distribution by the model. The term $E_{\theta\sim p(\theta|D)}\left[\log(p(\theta \mid D))\right]$ is a function of the data-generating process, not the model. Therefore, the minimization of KL divergence comes down to the maximization of the term $E_{\theta\sim p(\theta|D)}\left[\log(q(\theta))\right]$, which is the cross-entropy $H(p(\theta \mid D), q(\theta))$ of the distribution represented by the model and the true distribution of the data-generating process [43].

Note that up to this point, the model is deterministic, meaning that multiple forward passes of the model, given an input, will produce a single value for $\hat{\mu}$ and $\hat{\sigma}^2$. This also implies that a deterministic model is sufficient to give an estimate of data uncertainty. In the following section, it will be shown how this approach can be generalized to variational models, where multiple forward passes generate different predictions, and how this helps to quantify model uncertainty.

3.2.3 Epistemic (Model) Uncertainty

As stated earlier, Bayesian Neural Networks (BNNs) aim at inferring the probability distribution over the model parameters. At inference time, the values of the weights are drawn from the corresponding distributions. Since computing the precise posterior distributions of model weights that characterize model uncertainty is intractable, approximate variational techniques are used instead [59]. One of the most prominent methods in this category is approximate variational inference by Monte Carlo (MC) Dropout [39]. In this method, the neural network is trained by applying dropout to hidden layer nodes with a Bernoulli distribution. Then, at inference time, dropout is again applied randomly to the hidden layer nodes, resulting in variations in the weight matrices, hence in the predicted values. Dropout is shown to be mathematically equivalent to an approximation of Bayesian variational inference [39].

The inference process boils down to sampling T weights $\hat{\theta}_t$ from the approximate posterior $q(\theta)$ and applying Monte Carlo integration to get the predictive distribution [38]:

$$\begin{aligned} p(y^* \mid x^*, D) &= \int p(y^* \mid x^*, \theta) p(\theta \mid D) d\theta \\ &\approx \int p(y^* \mid x^*, \theta) q(\theta) d\theta \\ &\approx \frac{1}{T} \sum_{t=1}^{T} p(y^* \mid x^*, \hat{\theta}_t) = q(y^* \mid x^*) \end{aligned} \tag{3.8}$$

This process is referred to as Monte Carlo Dropout [39]. Note that with a Gaussian prior, this corresponds to combining multiple Gaussian distributions $\{N(\mu_{\hat{\theta}_t}, \sigma^2_{\hat{\theta}_t})\}_{t=1}^{T}$ into one $N(\hat{\mu}, \hat{\sigma}^2)$. Given that each draw of model weights results in two estimated parameters, we have a set of parameter estimates $\{[\hat{\mu}_t, \hat{\sigma}^2_t]\}_{t=1}^{T}$.

The predicted mean will be given by:

$$\hat{\mu} = E_{y \sim p(y|x^*, D)}[y] \approx \frac{1}{T} \sum_{t=1}^{T} \hat{\mu}_t \tag{3.9}$$

The total predictive variance will be given by [78]:

$$\hat{\sigma}^2 = \mathrm{Var}_{y \sim p(y|x^*, D)}[y] = E[\mathrm{Var}(y \mid x^*, D)] + \mathrm{Var}(E[y \mid x^*, D]) \tag{3.10}$$

$$\approx \underbrace{\frac{1}{T}\sum_{t=1}^{T}\hat{\sigma}_t^2}_{\text{Data uncertainty}} + \underbrace{\frac{1}{T}\sum_{t=1}^{T}\hat{\mu}_t^2 - \left(\frac{1}{T}\sum_{t=1}^{T}\hat{\mu}_t\right)^2}_{\text{Model uncertainty}} \tag{3.11}$$

The first term in Equation (3.11), $E[\text{Var}(y \mid x^*, D)]$, represents the uncertainty inherent to the data, which is not reducible. The second term, $\text{Var}(E[y \mid x^*, D])$, represents the uncertainty rooted in the model parameters. Note that the second term reduces to zero when the estimates $\{\hat{\mu}_t \mid t \in \{1, \ldots, T\}\}$ are identical, which means there is no disagreement between the models defined by the sampled weights.

For capturing model uncertainty, other techniques such as Deep Ensembles [85] can also be used. The Deep Ensembles approach trains multiple models with different initializations and combines their predictions by averaging them. Consequently, the collective size of the models and the training time grows linearly with the number of models in the ensemble. However, the Dropout method has a negligible model size and training time overhead since only one model is trained.

Note that a basic model can be extended to support Dropout Variational Inference simply by adding dropout layers, which do not add to the number of model parameters. This means the only training overhead is drawing a value from a Bernoulli distribution for each node in the forward path and backpropagation. Once a single model is trained, an unlimited number of predictions can be obtained for a given sample by different invocations

of the model as needed. Therefore, this study uses the Dropout method due to its simplicity, scalability, and very low training overhead, while benefiting from the mathematically grounded framework of BNNs for reasoning about model uncertainty.

Chapter 4

Theoretical Framework: Robust Plan Evaluation based on Approximate Probabilistic Machine Learning

This chapter presents a comprehensive theoretical framework for robust query optimization (Roq) based on approximate probabilistic machine learning. Building upon the concepts examined in the literature review, it systematically formalizes the notion of robustness in query optimization through three interconnected sections. Section 4.1 investigates the sources of uncertainty in plan evaluation and selection, providing precise definitions for plan robustness and cost model robustness while establishing design guidelines for robust

cost models. Section 4.2 introduces principled quantification measures for plan risk, estimation risk, and suboptimality risk, specifically tailored for ML-based cost models. Section 4.3 presents novel risk-aware plan evaluation and selection strategies, along with corresponding algorithms that leverage these risk measures to ensure robustness in practical implementations. Together, these sections establish a rigorous mathematical foundation for the architectural and experimental components presented in subsequent chapters.

4.1 Studying and Modeling Robustness

The uncertainty of a plan cost estimate is rooted in two main sources: a) the plan structure and the data that flow through the plan operators and b) the limitations of the cost model used to evaluate the plan.

4.1.1 Plan and Cost Estimation Uncertainty Modeling

The robustness of a plan can be formalized based on the behavior of its cost given various parameter values [154]. The cost of a plan is usually represented by a Parametric Cost Function (PCF) that shows its cost at different points in a multidimensional parameter space, which includes all combinations of values of different error-prone parameters that impact the cost of a plan. Previous works have suggested various properties of the PCF to

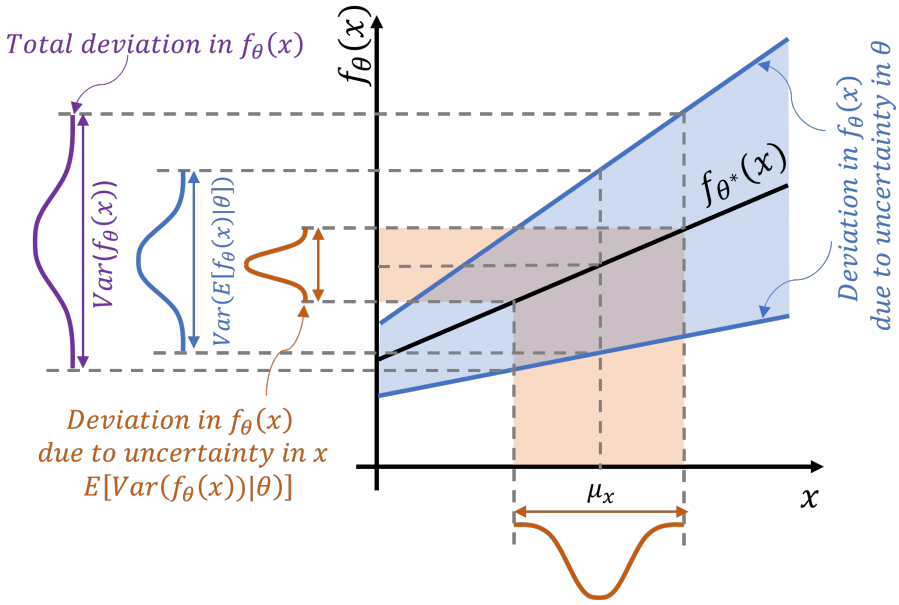


Figure 4.1: Cost Model Uncertainty Decomposition

measure plan robustness. For example, the slope of the PCF and its area under the curve are proposed as such measures [154].

We formalize this notion by modeling the uncertainty in the PCF and how it can be decomposed into uncertainties from various sources, including the structure of the plan and the parameters of the cost estimation model. Consider a PCF $f_\theta(\mathcal{X})$ with error-prone input parameters $\mathcal{X} = \{x_1, ..., x_n\}$, and a set of model parameters θ. The uncertainty in the estimation provided by $f_\theta(\mathcal{X})$ can be traced back to the uncertainty in a) input parameter estimates $\mathcal{X}$ and b) model parameters θ. This uncertainty ($\text{Var}(f_\theta(\mathcal{X}))$) can be decomposed using the law of total variance:

$$\text{Var}(f_\theta(\mathcal{X})|\mathcal{X} = \mathcal{X}^*) = E[\text{Var}(f_\theta(\mathcal{X})|\mathcal{X}^*, \theta)] + \text{Var}(E[f_\theta(\mathcal{X})|\mathcal{X}^*, \theta]) \tag{4.1}$$

where $\mathcal{X}^*$ represents the estimated distribution of the input parameters.

Theorem 4.1. *The term $E[Var(f_\theta(\mathcal{X})|\mathcal{X}^*, \theta)]$ represents the uncertainty rooted in the plan structure and its sensitivity to cardinality misestimations and the term $Var(E[f_\theta(\mathcal{X})|\mathcal{X}^*, \theta])$ represents the uncertainty rooted in model parameters.* □

Proof. Let us consider a cost model that is linear with respect to the size of a single error-prone cardinality (Figure 4.1):

$$f_\theta(x) = ax + b; \quad \theta = \{a, b\} \tag{4.2}$$

where a and b have normal errors: $a \sim \mathcal{N}(\mu_a, \sigma_a^2)$, and $b \sim \mathcal{N}(\mu_b, \sigma_b^2)$, and $x = x^* \sim \mathcal{N}(\mu_{x^*}, \sigma_{x^*}^2)$ is the approximated cardinality distribution for the input plan. The expected value $E[f_\theta(x)|x = x^*]$ can be computed a follows:

$$
\begin{aligned}
E[f_\theta(x)|x = x^*] &= E[a.x^* + b] = E[a.x^*] + E[b] \\
&= E[a].E[x^*] + E[b] \text{ (a and x are independent)} \\
&= \mu_a.\mu_{x^*} + \mu_b
\end{aligned}
\tag{4.3}
$$

The total variance $\text{Var}(f_\theta(x)|x = x^*)$ can be decomposed using the law of total uncertainty:

$$
\text{Var}(f_\theta(x)|x = x^*) = E[\text{Var}(f_\theta(x)|x^*, \theta)] + \text{Var}(E[f_\theta(x)|x^*, \theta]) \tag{4.4}
$$

The first and the second components can be computed as follows:

$$
\begin{aligned}
E[\text{Var}(f_\theta(x)|x^*, \theta)] &= E[\text{Var}(ax + b|x^*, a, b)] = E[a^2\text{Var}(x^*)] \\
&= E[a^2\sigma^2_{x^*}] \\
&= \sigma^2_{x^*}E[a^2]; a^2 \sim \chi^2(\text{d.o.f} = 1) \\
&= \sigma^2_{x^*}(\mu^2_a + \sigma^2_a)
\end{aligned}
\tag{4.5}
$$

Equation 4.5 captures the variance that is influenced by the error in cardinality x^* and the sensitivity of the cost model to such errors. This component has a direct relationship with the uncertainty in the input data ($\sigma^2_{x^*}$) on the one hand and the mean and variance of the slope (i.e. the first derivative) ($\mu^2_a + \sigma^2_a$) of $f_\theta(x)$ on the other. The term $\mu^2_a +$

σ_a^2 determines the level of sensitivity to cardinality error. *Therefore, mapping the term* $E[Var(f_\theta(x)|x^*, \theta)]$ *to the robustness of the plan is justified as it captures both the error rooted in the input cardinality and the sensitivity of the plan to this error.* Note that this component is reduced to zero if $\sigma_{x^*}^2 = 0$ or $\mu_a^2 + \sigma_a^2 = 0$.

$$\begin{aligned} \text{Var}(E[f_\theta(x)|x^*, \theta]) &= \text{Var}(E[a.x + b|x^*, a, b]) = \text{Var}(a.E[x^*] + b) \\ &= \text{Var}(a.\mu_{x^*} + b) \\ &= \mu_{x^*}^2.\text{Var}(a) + \text{Var}(b) + 2\text{Cov}(a.\mu_{x^*}, b) \\ &= \mu_{x^*}^2.\sigma_a^2 + \sigma_b^2 + 2\mu_{x^*}.\text{Cov}(a, b) \end{aligned} \tag{4.6}$$

Equation 4.6 captures the uncertainty rooted in the model parameters σ_a^2 and σ_b^2, and their covariance $\text{Cov}(a, b)$. This is the uncertainty rooted in the limitations of the cost model. *Therefore, mapping the term* $Var(E[f_\theta(x)|x^*, \theta])$ *to the uncertainty rooted in model parameters is justified.* Note that the term $\text{Var}(E[f_\theta(x)|x^*, \theta])$ is reduced to zero if and only if $\sigma_a^2 = 0$, $\sigma_b^2 = 0$, and $\text{Cov}(a, b) = 0$. In such a scenario, the total variance of the plan's cost estimate will be determined by the cardinality errors and the sensitivity of the cost to those errors.

Note that the assumption of a normal distribution for a, b, and x^* is made to facilitate the discussion and that it is inconsequential to the conclusions. The same applies to the

assumption of linearity. These assumptions are discussed in more detail in the following sections. □

The decomposition provided in Theorem 4.1 is illustrated in Figure 4.1, with a simple linear cost model with one error-prone cardinality parameter. The first term ($E[\text{Var}(f_\theta(\mathcal{X})|\mathcal{X}^*, \theta)]$) has a direct relationship with a) the error in the input cardinality and b) the slope of $f_\theta(\mathcal{X})$ which determines its sensitivity to the error in the input. Therefore, its quantification is a suitable measure for *plan robustness*. The second term ($\text{Var}(E[f_\theta(\mathcal{X})|\mathcal{X}^*, \theta])$), on the other hand, is directly influenced by the uncertainty in the model parameters, and it is a suitable measure for *model robustness*. This represents the uncertainty rooted in our lack of knowledge about an ideal cost model.

On the Normality Assumption: The normality assumption in the uncertainty decomposition is justified for two reasons. First, input parameters of the cost model and the outputs can be transformed using Box-Cox, logarithmic, or other power transformations to approximately conform to a normal distribution. This is demonstrated in the experimental study (Section 6), where the impact of such transformations on target distributions is illustrated in Figure 4.2 for the JOB benchmark. Avoiding such transformations would necessitate modeling arbitrary distributions, making any reasoning about uncertainty modeling impractical. Second, the key insight of the uncertainty decomposition—that uncer-

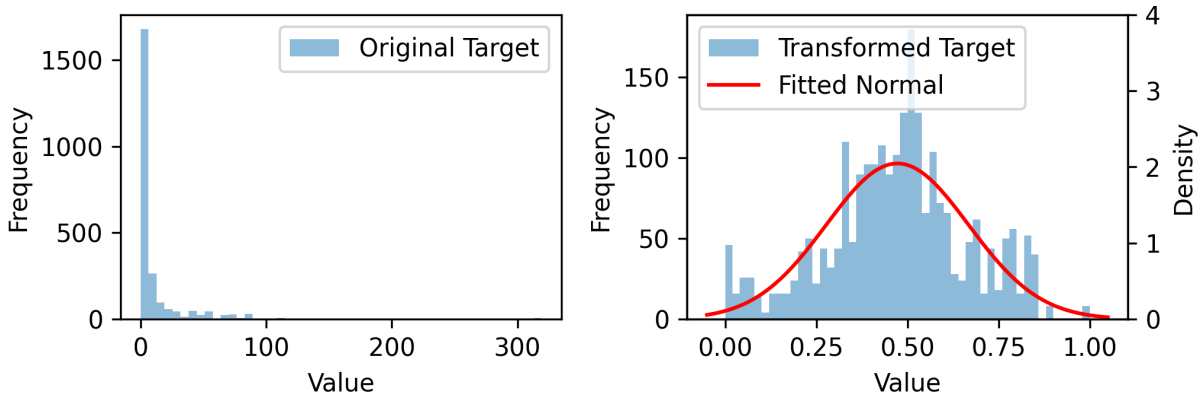


Figure 4.2: Target distributions before and after transformation

tainty can be separated into plan-structure-related and model-parameter-related components—holds regardless of the specific distributional assumptions. The normal distribution provides a reasonable approximation that enables analytical solutions while maintaining the validity of the decomposition principle.

On the Linearity Assumption: The linearity assumption in the proof is also inconsequential to the validity of the uncertainty decomposition. Even when the cost model is non-linear, it can be approximated as a linear function in the close vicinity of a cardinality estimate point, provided that the function is differentiable. This local linearization is guaranteed by Taylor's theorem, which states that any differentiable function can be approximated by its first-order Taylor expansion in a sufficiently small neighborhood around any point. The key assumption here is differentiability, which is reasonable for cost models since they typically involve smooth mathematical operations such as arithmetic, expo-

nentials, and other continuous functions. In query optimization, cardinality estimates are typically within some reasonable range of the true values, making the "local neighborhood" assumption realistic for practical purposes. The linear approximation effectively captures the sensitivity (gradient) of the cost function to parameter changes, which is the core insight needed for robustness analysis. This approach is widely used in sensitivity analysis across various fields, where complex non-linear systems are analyzed through local linearization. The fundamental principle of the uncertainty decomposition—separating the effects of input parameter uncertainty from model parameter uncertainty—remains valid regardless of the specific functional form of the cost model, as long as it is differentiable.

4.1.2 What Determines Plan Robustness?

Plan robustness can be formalized using the properties of a plan which are determined by the behavior of its cost given various parameter values [154]. The cost of a plan is usually represented by a Parametric Cost Function (PCF) that has different values at different points in the parameter space. The parameter space includes all combinations of values of different error-prone parameters that impact the cost of a plan. A robust plan is defined as one with limited worst-case performance degradation [52]. This property can be characterized by the maximum Suboptimality, which is defined as follows.

Definition 4.1. ***Suboptimality*** *of the plan p_i is the ratio of its execution cost ($ET(\cdot)$)*

to the cost of the optimal plan p_0:

$$Subopt(p_i) = \frac{ET(p_i)}{ET(p_0)}, \quad where\ 1 \leq Subopt(p_i) < \infty \tag{4.7}$$

The maximum of this ratio over the parameter space of the estimated cost determines the maximum suboptimality (MSO) and is suggested as a measure of robustness [32]. This measure is illustrated in Figure 4.3. Let us assume that the cardinality parameter can take a value in the range $[f_{\downarrow}, f_{\uparrow}]$. At each point, the suboptimality of each plan compared with the optimal is computed. MSO_{rob} and MSO_{vol} represent the maximum of this ratio for PCF_{rob} and PCF_{vol} respectively. Since $\text{MSO}_{\text{rob}} < \text{MSO}_{\text{vol}}$, the plan represented by PCF_{rob} is considered to be more robust in this range.

Guideline 4.1. *Roq focuses on the suboptimality of the selected plan with respect to the optimal plan. The tail-end of the distribution of suboptimality characterizes the worst-case performance of the optimization method and can be used as a measure of robustness. We use this measure to evaluate the robustness of the plan evaluation and selection strategies.*

4.1.3 The Impact of Plan Structure on Robustness

The structure of a plan can affect the sensitivity of its execution cost to cardinality estimation errors, and hence its robustness. We demonstrate this phenomenon with an example

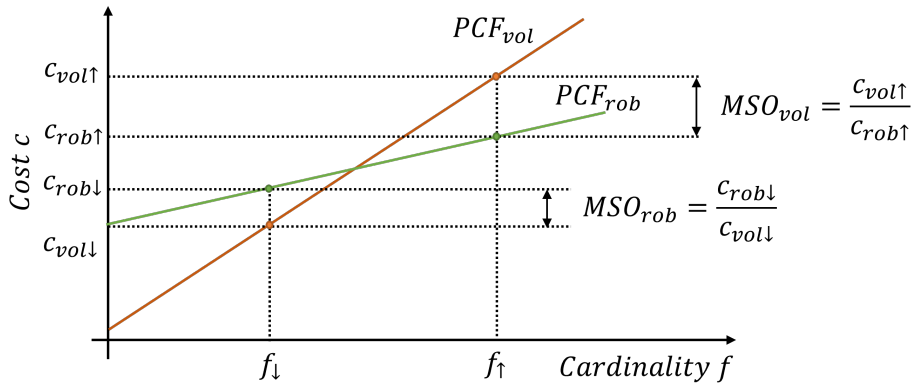


Figure 4.3: the PCF of two plans with respect to a cardinality value highlighting the maximum suboptimality as a measure of plan robustness

that highlights the impact of join operators on robustness.

Example 4.1. *Different join algorithms have different levels of sensitivity to cardinality estimation errors. Let us assume the following simple cost model that models the cost of Nested-loops joins and Hash joins based on the cardinality of the base tables:*

$$Cost(S \bowtie R) = \begin{cases} |S| + |S| \cdot |R| & if \ \bowtie = \bowtie^{NL} \\ (1+\lambda) \cdot (|S| + |R|) & if \ \bowtie = \bowtie^{HS} \end{cases} \tag{4.8}$$

Where $|X|$ denotes the cardinality of table X, NL denotes Nested-loops join, and HS denotes Hash join. A Nested-loops join scans the outer table (S) once. For every row of the outer, it scans all rows of the inner table (R) and returns the matching rows. A Hash join, however, assuming R fits in memory and no partitioning is needed, scans the inner

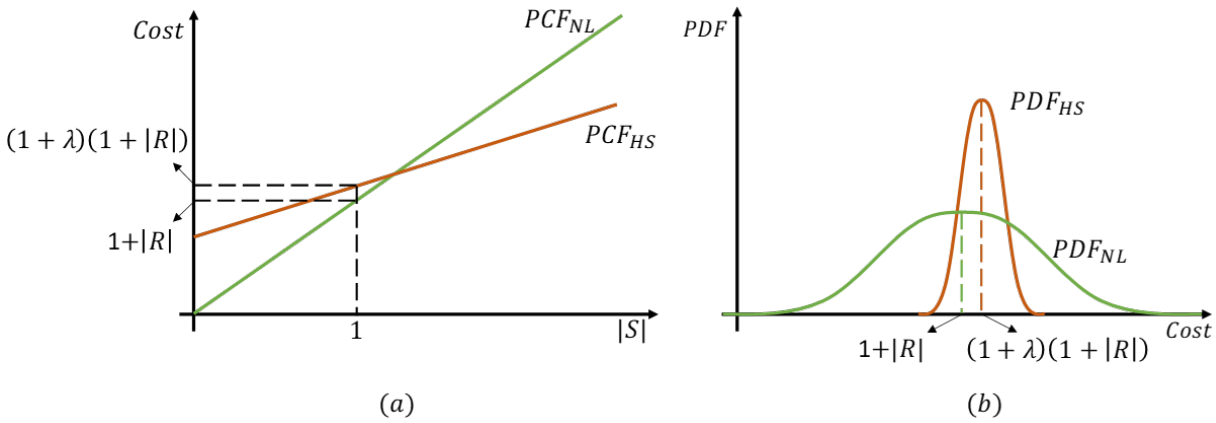


Figure 4.4: (a) Cost with respect to the cardinality of the outer table around $|S| = 1$, (b) PDF for the cost when $E[|S|] = 1$ for a plan with Nested-loops join vs. Hash join

once and builds a hash table based on the join column(s). It then scans and hashes S once; for every row in the outer, it probes the inner hash table and returns the matching rows. The hashing overhead per row is denoted by λ. *Figure 4.4a shows the cost models of these two join algorithms when plotted as a PCF with respect to the cardinality of the outer (*$|S|$*).*

Given the overhead involved for hashing in the Hash join (λ*), when* $|S|$ *is expected to equal one (*$E[|S|] = 1$*), it appears to be more expensive than the Nested-loops join. But when* $|S|$ *is larger than one, the cost of the Nested-loops join grows rapidly, making it suboptimal. Figure 4.4b shows the probability density functions (PDF) of the two plan costs when* $E[|S|] = 1$. *However, the cost of the plan with a nested-loops join grows rapidly with the size of the outer relation (*$|S|$*). Let us assume* $|R| = 100$ *and* $\lambda = 1$, *and that the cardinality estimate* $|S|$ *is one order of magnitude off (*$E[|S|] = 10$*). Then*

$PCF_{NL}(|S| = 10, |R| = 100) = 1010$ while $PCF_{HS}(|S| = 10, |R| = 100) = 220$, which means the actual execution cost of the nested-loops join will be 4.5x larger than the hash join. Therefore, in this example, the hash join plan is more robust to the cardinality misestimation. This example intuitively demonstrates the validity of the PCF's slope as a measure of robustness. □

This example demonstrates that some join operators are more sensitive to cardinality misestimations than other join operators. In addition, based on well-established prior research findings, the magnitude of cardinality estimation errors grows exponentially with the number of joins [61]. Based on these facts, we have the following observation.

Observation 4.1. *Plans that use sensitive join operators closer to the root of a deep plan tree may not be robust to cardinality errors.*

The is just one example of the role of plan structure on robustness. More examples can be provided, which is beyond the scope of this thesis. As demonstrated earlier, this aspect of robustness is captured by the term $E[\text{Var}(f_\theta(\mathcal{X})|\mathcal{X}^*, \theta)]$ in Equation 4.1.

4.1.4 The Impact of the Cost Model on Robustness

Cost models provide approximations for the amount time and/or resources required to process a plan. These approximations are inherently erroneous because of the limitations

of the cost model. In general, unrealistic assumptions lead to designing imperfect models with erroneous parameters. The uncertainty rooted in erroneous model parameters is captured by the term $\text{Var}(E[f_\theta(\mathcal{X})|\mathcal{X}^*, \theta])$ in Equation 4.1. Depending on how the cost model is built, imperfections of the cost model have different sources. In the following sections, we discuss these imperfections for classical and ML-based cost models.

Robustness of Classical Cost Models The classical cost models are built by modeling the behavior of various plan operators in the way they process records and need computing resources including I/O operations, memory, and CPU. They make assumptions about the computing resources available at runtime that may not hold in practice. In addition, these cost models heavily rely on the accuracy of the cardinality estimates that approximate the amount of data processed through each operator. Cardinality estimates are in turn erroneous due to simplifying assumptions made about the distribution of the underlying data, as well as pairwise correlations between different database attributes. Furthermore, classic cost models evaluate each plan operator in isolation. They are inherently incapable of accounting for the position of operators in the plan and do not take into account the upstream operators leading to the target operator. Therefore, classic cost models do not have the capability to account for cardinality error propagation through plan operators [61].

Observation 4.2. *Given the structural characteristics of a plan that impact robustness,*

such as the one provided in Observation 4.1, a robust cost model must be capable of ingesting the structural information of the plan in order to account for error propagation patterns.

Therefore, classic cost models are not robust because a) they rely on many simplifying assumptions, b) they are sensitive to errors in the input parameters such as cardinality estimates, and c) they are incapable of accounting for error propagation patterns.

Robustness of ML-based Cost Models There are recent research works using ML to learn the cost directly from latencies collected at runtime, e.g. [57, 103, 135]. The learned cost models have an inherent ability to reduce the reliance on noisy input features. To this end, they rely on the ability of ML models to find the best weights that collectively determine its reliance on any input feature. With a high inaccuracy in some input parameters, the model learns to reduce their impact on the cost estimates. Thus, the learned models reduce the sensitivity to noisy input features, making them more robust to errors. This inherent capability was previously demonstrated by Neo [101]. In addition, ML-based approaches typically make fewer simplifying assumptions, if any. Furthermore, they evaluate a query execution plan as a whole, taking into account the structural information of the plan. Therefore, they have the capability to learn error propagation patterns and reduce sensitivity of the final estimate to such errors. From these perspectives, learned cost models may be considered more robust than classical cost models. However, ML-based approaches pose a new threat to robustness: the threat of bias and lack of generalization. Due

to various factors involved in ML training (e.g. limited coverage of the training data, lack of sufficient inductive bias in model architecture, overfitting), the predictions on unseen queries or plans may have significant inaccuracies. This gap necessitates new approaches that quantify uncertainties due to training data coverage and the limitations of the model.

Based on the above discussion, we propose the following definition for a robust cost model, followed for a guideline that informs this research.

Definition 4.2. *A robust cost model a) is not sensitive to errors in the input parameter estimates, b) is capable of accounting for the plan structure, c) does not rely on simplifying assumptions, and d) quantifies the estimation uncertainties.*

Guideline 4.2. *Learned cost models are more promising to support robustness, for three reasons: a) they learn the cost based on input data and not based on a-priori rigid assumptions, b) they are inherently capable of learning to reduce sensitivity to uncertain input features [101], and c) they are capable of ingesting the whole plan and taking into account its structural characteristics. In Roq, we take a different route than the state-of-the-art ML-based query optimizers and design a novel learned cost model that estimates risks related to plans and their estimated costs.*

4.2 Ensuring Robustness via Risk Quantification

We recognized the probabilistic nature of cost estimates and provided guidelines for the creation of a robust cost model based on ML. We continue with the definition of quantification measures of the 3 types of risks, which will be employed to ensure robustness. The proposed quantification is tailored to an ML-based cost model and employs approximate probabilistic ML. We describe an example with scenarios of plan cost uncertainties that is used in the discussion of the proposed measures.

Example 4.2. *Let us have another look at Figure 1.3 from chapter 1. This figure illustrates scenarios for plan cost inaccuracies. Each plot shows the probability density function (PDF) of the estimated cost of two alternative plans. The costs are assumed to have a normal distribution. In each case, assuming the optimizer estimates the most likely cost for each plan, the plan with minimum expected cost is considered as the optimal one. The alternative plan, while having a higher expected cost, is also likely to be optimal at runtime with a non-zero probability. Note that the best-case scenario for the optimizer is when the plan costs are consistently accurate, as the optimizer naively assumes (Figure 1.3a). In such a case, the likelihood of plan X being more expensive than plan Y is low. More likely, however, the estimates have various levels of inaccuracies. Figure 1.3b/c/d represent such scenarios. If one of the two estimates is relatively more inaccurate, the risk can increase substantially*

(Figure 1.3b/c). The worst-case is if both estimates are relatively inaccurate (Figure 1.3d), as the risk of choosing the expected optimal plan increases. This example highlights the importance of taking into account the PDFs of the cost estimates rather than only the expected values. □

4.2.1 Uncertainty Modeling in ML

The uncertainties captured in Equation 4.1 depend on the marginal distribution

$$p(f_\theta(\mathcal{X})|\mathcal{X}^*, \theta) \tag{4.9}$$

This distribution is conditioned on the cost model parameters and the input parameters. In the context of learned cost models, the model parameters are obtained through training based on labeled samples $\mathcal{D} = \{(x_i, y_i)|\, \forall i \in \{1, \ldots, n\}\}$. Therefore, the distribution must be conditioned on the training samples $\mathcal{D}$: $p(f_\theta(\mathcal{X})|\mathcal{X}^*, \mathcal{D})$. Making inferences through this distribution comes down to evaluating the following equation and is called Bayesian inference [59]:

$$p(f_\theta(\mathcal{X})|\mathcal{X}^*, \mathcal{D}) = \int p(f_\theta(\mathcal{X})|\mathcal{X}^*, \theta) p(\theta|\mathcal{D})\, d\theta \tag{4.10}$$

The marginal distribution $p(\theta|\mathcal{D})$ represents the uncertainty in the parameters of the model given the training samples $\mathcal{D}$. This term captures the additional variance existing in a learned model compared with an analytical (classical model), and is caused by the stochastic nature of the training process. Therefore, the decomposition of the total variance provided in Equation 4.1 can be rewritten as follows:

$$\text{Var}(f_\theta(\mathcal{X})|\mathcal{X} = \mathcal{X}^*) = E[\text{Var}(f_\theta(\mathcal{X})|\mathcal{X}^*, \mathcal{D})] + \text{Var}(E[f_\theta(\mathcal{X})|\mathcal{X}^*, \mathcal{D}]) \tag{4.11}$$

The first term in Equation 4.11 is aleatoric (or data) uncertainty and the second term is epistemic (or model) uncertainty. The variability of the behavior of the real-world phenomena is explained by data uncertainty. Data uncertainty can also be caused by noisy input vectors or labels, as well as low-dimensional input vectors that do not adequately explain the sample [59]. Model uncertainty, however, captures the variability of predictions due to a lack of knowledge about an ideal model. This is rooted in limitations in model architecture, training procedure, or the coverage of the training data [40]. For a learned cost model, these notions can be quantified using various techniques from the ML literature [40]. In the following sections, we explain the approaches we take to quantify these two types of uncertainty.

4.2.1.1 Data Uncertainty

A neural network can be designed to predict the parameters of probability distribution [119]. In other words, it not only predicts the conditional expected value but also the conditional variance of the target given the training data and the input sample ($E[Var(y|x^*, D)]$). This is done by adding a second branch to the output of the neural network, which predicts the variance. Optimizing the following loss function corresponds to maximizing the log-likelihood with a Gaussian prior as shown in [119]:

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\ln(\sigma_i^2)}{2} + \frac{(y_i - \mu_i)^2}{2\sigma_i^2} + \frac{1}{2}\ln 2\pi\right) \tag{4.12}$$

where μ_i is plugged in from the first branch of the output layer, σ_i^2 from the second one, and y_i from the labels.

4.2.1.2 Model Uncertainty

Model uncertainty is captured by the variance of the conditional expected values of the target ($Var(E[(y|x^*, D)])$). To capture this, we use approximate variational inference by Monte Carlo (MC) Dropout [39]. In this method, the neural network is trained by applying dropout on hidden layers with a Bernoulli distribution. At inference time, dropout is again

applied randomly to the hidden layers, resulting in variations in the weight matrices, hence in the predicted values. Note that with a Gaussian prior, this corresponds to combining multiple Gaussian distributions $\{N(\mu_{\hat{\theta}_t}, \sigma^2_{\hat{\theta}_t})\}_{t=1}^T$ into one $N(\hat{\mu}_t, \hat{\sigma}^2_t)$. Given each draw of model weights results in two estimated parameters, we have a set of parameter estimates $\{(\hat{\mu}_t, \hat{\sigma}^2_t)\}_{t=1}^T$. The predicted mean is:

$$\hat{\mu} = E_{y\sim p(y|x^*,D)}[y] \approx \frac{1}{T}\sum_{t=1}^{T}(\hat{\mu}_t) \tag{4.13}$$

The data, model, and total uncertainties are captured by the following equations respectively [78]:

$$\hat{\sigma}^2_d = E[Var(y|x^*, D)] = \frac{1}{T}\sum_{t=1}^{T}(\hat{\sigma}^2_t) \tag{4.14}$$

$$\hat{\sigma}^2_m = \text{Var}\left(E[(y|x^*, D)]\right) = E\left[E\left[(y|x^*, D)\right]^2\right] - \left(E\left[(y|x^*, D)\right]\right)^2$$

$$= \frac{1}{T}\sum_{t=1}^{T}\hat{\mu}^2_t - \left(\frac{1}{T}\sum_{t=1}^{T}\hat{\mu}_t\right)^2 \tag{4.15}$$

$$\hat{\sigma}^2 = \hat{\sigma}_d^2 + \hat{\sigma}_m^2 \tag{4.16}$$

4.2.2 Plan Risk

As explained in Section 4.1, data uncertainty captures the uncertainty rooted in input parameters and the sensitivity of the plan to those uncertainties. Therefore, we relate the riskiness (or robustness) of a plan to data uncertainty and quantify it by the function provided in Equation 4.14. Consider the optimal plan in Figure 1.3c. Let us assume the variance of the distributions represents the data uncertainty (i.e., the model uncertainty is zero). Although the expected cost of this plan is lower than that of the alternative plan, its potential deviation at runtime is larger. Running this plan in various conditions and in presence of different parameter values, is likely to lead to longer execution times in some scenarios. The cost for the alternative plan, however, has a tighter distribution while having a higher expected cost. In this example, the parameter representing the deviation of the cost distribution can be considered as the risk inherent to the plan itself.

4.2.3 Estimation Risk

Estimates are inherently subject to errors. Therefore, regardless of how robust a plan is, the estimate of its cost is also subject to uncertainties. We use as a measure for the Estimation Risk the model uncertainty as quantified using the function in Equation 4.15. The scenario represented in Figure 1.3c can be studied assuming that the deviation represents model uncertainty (i.e., data uncertainty is zero). In such a case, the deviation of the cost from the expected value quantifies the uncertainty rooted in the modeling limitations. Therefore, a plan with a wider distribution will have a higher estimation risk.

4.2.4 Suboptimality Risk

As stated in the problem formulation with Approach 1, the goal of robust query optimization is to find a plan with minimum risk of suboptimality. This can be formalized as the likelihood of the plan being suboptimal compared with the alternative plan(s). To explain this, let us revisit Example 4.2. We assume that the cost of the optimal plans in all scenarios is 8, and it is 10 for the alternative plans. The difference between the scenarios is the standard deviation of the probability distributions. Let us say the cost for the optimal plan (plan X) is denoted by $C(X) \sim \mathcal{N}(\mu_x, \sigma_x^2)$ and the cost for the alternative plan (plan Y) is denoted by $C(Y) \sim \mathcal{N}(\mu_y, \sigma_y^2)$. Also, let us assume the covariance between $C(X)$ and

$C(Y)$ is denoted as $\text{Cov}(x, y)$. Then the probability of the alternative plan being cheaper than the optimal plan can be denoted by:

$$R(p_X, p_Y) = P(C(X) - C(Y) > 0) \tag{4.17}$$

Where $R(p_X, p_Y)$ is the risk of picking X over Y. Note that:

$$D = C(X) - C(Y) \sim \mathcal{N}(\mu_x - \mu_y, \sigma_x^2 + \sigma_y^2 - 2\text{Cov}(x, y)) \tag{4.18}$$

For the sake of simplicity, let us assume X and Y are independent variables, therefore $\text{Cov}(x, y) = 0$. Then:

$$D = C(X) - C(Y) \sim \mathcal{N}(\mu_x - \mu_y, \sigma_x^2 + \sigma_y^2) \tag{4.19}$$

The probability of $C(X) - C(Y) > 0$ is then obtained by computing the z-score for $C(X) - C(Y) = 0$:

$$\text{zscore} \approx \frac{0 - (\mu_x - \mu_y)}{\sqrt{\sigma_x^2 + \sigma_y^2}} = \frac{\mu_y - \mu_x}{\sqrt{\sigma_x^2 + \sigma_y^2}} \tag{4.20}$$

In all 4 scenarios, $\mu_x = 8$, $\mu_y = 10$, and $\mu_y - \mu_x = 2$. Table 4.1 outlines the computation

Table 4.1: Computation of risks associated with selecting the optimal plan at different uncertainty levels

Scenario	$\boldsymbol{\sigma_x}$	$\boldsymbol{\sigma_y}$	$\sqrt{\boldsymbol{\sigma_x^2 + \sigma_y^2}}$	**z-score**	**P(C(X)-C(Y)> 0)**
a	1	1	$\sqrt{2}$	≈ 1.41	$\approx 7.9\%$
b	1	4	$\sqrt{17}$	≈ 0.49	$\approx 31.6\%$
c	4	1	$\sqrt{17}$	≈ 0.49	$\approx 31.6\%$
d	4	4	$\sqrt{32}$	≈ 0.35	$\approx 36.3\%$

of the risk associated with choosing the optimal plan for all scenarios illustrated in Figure 1.3.

Note that the computed risks are consistent with the qualitative study of the graphs in Figure 1.3. These computations can be generalized to compute the risk for choosing the optimal plan in the presence of several alternative plans to give the Suboptimality Risk. This can be computed as the average risk of picking the target plan over any other plan in the search space, as follows:

$$\text{SOR}(p_i) = \frac{1}{S-1} \sum_{j=1, j \neq i}^{S} R(p_i, p_j) \tag{4.21}$$

Where S is the total number of plans in the search space, $\text{SOR}(p_i)$ is Suboptimality Risk for plan p_i in the presence of every other $S-1$ plan, and $R(p_i, p_j)$ is the risk of picking the plan p_i over plan p_j. Note that this risk factor can be computed using either model or data uncertainty for σ values. The former gives the risk rooted in modeling limitations while the latter gives the risk rooted in the plan structure. In Roq, we choose to compute

this value using the total uncertainty to account for both types of risk. We demonstrate in the following section that SOR(p_i) is a function of the expected cost and robustness of all plans in the search space.

4.2.5 Extended Discussion on the Formulation of SOR

To instantiate SOR with a Gaussian prior, let us rewrite Equation 4.19 as:

$$\delta = C_x - C_y \sim \mathcal{N}(\mu_\delta, \sigma_\delta^2)$$

where $\mu_\delta = \mu_x - \mu_y, \sigma_\delta^2 = \sigma_x^2 + \sigma_y^2$ The probability density function $\mathcal{F}(\delta)$ for a variable δ with normal distribution $\mathcal{N}(\mu_\delta, \sigma_\delta^2)$ is:

$$\mathcal{F}(\delta) = \frac{1}{\sqrt{2\pi\sigma_\delta^2}} e^{-\frac{(\delta-\mu_\delta)^2}{2\sigma^2}}$$

By substituting δ, μ_δ and σ_δ^2 this equation can be expanded as:

$$\mathcal{F}(\delta) = \frac{1}{\sqrt{2\pi(\sigma_x^2 + \sigma_y^2)}} e^{-\frac{(C_x - C_y - (\mu_x - \mu_y))^2}{2(\sigma_x^2 + \sigma_y^2)}}$$

The risk of suboptimality of plan x over plan y (i.e. $R(x, y)$) can be computed as:

$$R(x, y) = P(C_x > C_y) = P(\delta > 0) = \int_0^\infty \mathcal{F}(\delta) d\delta$$

where as demonstrated above, $\mathcal{F}(\delta)$ is a function of C_x, C_y, μ_x, μ_y, σ_x, and σ_y.

This demonstrates that SOR(p_i), which is the average risk of suboptimality of plan p_i as given in Equation 4.21, is a function of the expected value and variance of cost (and hence robustness) of all plans in the search space.

Computation Overheads In order to minimize the compilation overheads, the computation of SOR can be vectorized, such that at each step of comparing several plan fragments $R(p_i, p_j)$ is computed as a matrix $R_{n\times n}$ where n is the number of plans to be compared and $R_{i,j} = R(p_i, p_j)$. To this end, we define $D_{n\times n}$ where $D_{i,j} = \mu_i - \mu_j$ and $S_{n\times n}$ where $S_{i,j} = \sqrt{\sigma_i^2 + \sigma_j^2}$. Then the z-scores for various comparing different pairs of plans can be computed as:

$$Z_{n\times n} = -D_{n\times n} \oslash S_{n\times n} \tag{4.22}$$

Where the $\oslash$ symbol denotes the Hadamard (elementwise) division. The computation overheads using this vectorization approach are negligible as demonstrated in the experi-

mental study.

4.2.6 The Independence Assumption

Let us now revisit Equation 4.18 without assuming independence. Note that the covariance component can be computed as:

$$\text{Cov}(x, y) = E(C(X) \cdot C(Y)) - E(C(X)).E(C(Y))$$

An accurate estimation of the risk associated with selecting the plan X over the plan Y would require the probability distribution of the cost for each plan, as well as the pairwise joint distribution $C(X) \cdot C(Y)$. Assuming the independence between $C(X)$ and $C(Y)$ results in overestimating the risk by increasing the variance of $C(X) - C(Y)$. We assume independence for two reasons: a) the computation overheads would be impractical and computationally prohibitive and b) even with the independence assumption made, the risk quantification provides significant improvements. The effectiveness of the simplified quantification, as demonstrated in the experimental study, can be explained by the fact that the overestimation effect for all pairs of plans is consistent, effectively canceling each other out.

4.3 Risk-aware Plan Evaluation

The classic optimizers select a plan from a set of plans solely based on the expected cost. By doing so, they assume that the estimated expected costs have zero uncertainties. In contrast, when the uncertainties, and, thus, risk, are quantified, a new family of plan evaluation and selection methods can be devised that account for them. We propose three risk-aware plan evaluation strategies. While the first two strategies use risk measures directly for selecting plans, the third uses risk to prune the search space. We design algorithms that implement these strategies.

4.3.1 Plan Selection by Suboptimality Risk

Plan selection by suboptimality risk follows the first approach to the problem formulation where the problem is defined a risk minimization one. As such, the suboptimality risk is computed for every plan in the search space, and the one with the minimum risk is selected, based on the risk quantification method suggested in section 4.2. Given a set $\mathcal{P} = \{p_i | \forall i \in \{1, \ldots, S\}\}$ of S plans, it computes the suboptimality risk $\text{SOR}(p_i)$ for p_i. The plan with minimum $\text{SOR}(p_i)$ is selected. The algorithm can use either plan or estimation risk, or a combination of the two.

Algorithm 4.1 Plan Selection by SubOpt Risk $(\mathcal{P}, \mathcal{C})$

Inputs:
$\mathcal{P} = \{p_i \,|\, \forall i \in \{1, \ldots, S\}\}$ Set of plans to be evaluated
$\mathcal{C} = \{C_{p_i} \sim N(\mu_{p_i}, \sigma^2_{p_i}) \,|\, \forall i \in \{1, \ldots, S\}\}$ cost distributions

Output: Selected plan P^*
Initialize $P^* \leftarrow \emptyset$
Initialize $SOR \leftarrow \emptyset$ an empty set for subopt risks
for $p_i \in \mathcal{P}$ **do**
    for $p_j \in \mathcal{P}$ **do**
        if $j \neq i$ **then**
            $C_{p_i} - C_{p_j} \sim N(\mu_i - \mu_j, \sigma_i^2 + \sigma_j^2 - 2\text{Cov}(C_{p_i}, C_{p_j}))$
            $R(p_i, p_j) := P(C_{p_i} - C_{p_j} > 0)$
        end if
    end for
    $R(p_i) := \frac{1}{S-1} \sum_{j=1, j\neq i}^{S} R(p_i, p_j)$
    $SOR \leftarrow SOR \cup \{R(p_i)\}$
end for
$y^* := \arg\min_{i \in \{1,\ldots,S\}} SOR$
$P^* \leftarrow p_{y^*}$
return P^*

4.3.2 Conservative Plan Selection

The conservative plan selection strategy assumes that the plans have an execution cost that is higher than their estimated cost, and this difference is proportional to the standard deviation of their estimated cost. It eliminates risky plans (that is, plans with high standard deviation) that happen to be low cost. Given a set $\mathcal{P} = \{p_i | \forall i \in \{1, \ldots, S\}\}$ of S plans, where each p_i has a cost distribution $C(p_i) \sim \mathcal{N}(\mu_{p_i}, \sigma^2_{p_i})$, rather than picking the plan with minimum μ_{p_i}, it picks the plan with minimum $\mu_{p_i} + f_s \sigma_{p_i}$, where f_s is a parameter tuned

on a validation set. The algorithm can be specialized to use either the plan or the estimate risk, or both. This algorithm follows the second approach to the problem formulation where a linear combination of the two objectives $\alpha.\mu_{p_i} + (1-\alpha).\sigma_{p_i}$ is minimized, where $f_s = \frac{1-\alpha}{\alpha}$. As discussed in the introduction (Section 1), this formulation can be interpreted as optimizing a risk-adjusted execution time $m + f \cdot \sigma$, which represents an upper bound estimate of execution time with a certain confidence level under the normal distribution assumption.

To illustrate this concept, consider the example shown in Figure 4.5. We have two candidate plans: Plan A with mean execution cost $\mu_A = 100$ and standard deviation $\sigma_A = 40$, and Plan B with mean execution cost $\mu_B = 120$ and standard deviation $\sigma_B = 10$. Under traditional cost-based optimization, Plan A would be selected due to its lower average cost (100 vs 120). However, when applying conservative plan selection with $f_s = 1$, we compare the risk-adjusted costs: $\mu_A + f_s \cdot \sigma_A = 100 + 1 \cdot 40 = 140$ for Plan A, and $\mu_B + f_s \cdot \sigma_B = 120 + 1 \cdot 10 = 130$ for Plan B. The conservative strategy selects Plan B because its risk-adjusted cost (130) is lower than Plan A's (140), even though Plan A has a lower expected cost. This demonstrates how the conservative approach prioritizes plans with lower variability over those with lower average cost, aligning with the interpretation of $m + f \cdot \sigma$ as an upper bound estimate with approximately 84% confidence level when $f = 1$.

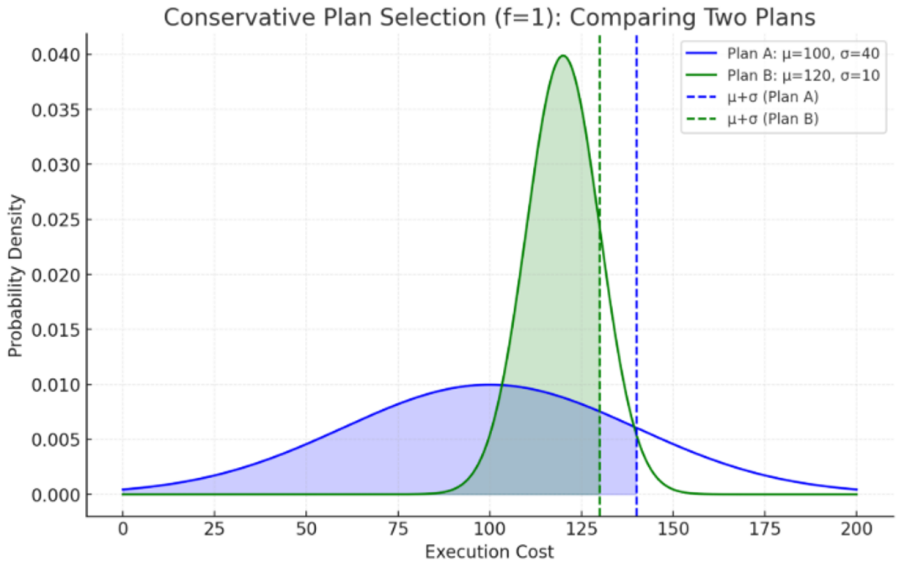


Figure 4.5: Conservative Plan Selection ($f = 1$): Comparing Two Plans. The figure illustrates execution cost distributions for two plans with different mean costs and variability. Plan A (blue) has $\mu = 100, \sigma = 40$ with $\mu + \sigma = 140$, while Plan B (green) has $\mu = 120, \sigma = 10$ with $\mu + \sigma = 130$. Despite Plan A having a lower average cost, Plan B is preferred under conservative selection ($f = 1$) due to its lower risk-adjusted cost estimate.

4.3.3 Search Space Pruning by Plan and Estimation Risks

This strategy prunes the search space to eliminate highly risky plans, based on either the plan or the estimation risks. Algorithm 4.3 is designed to implement this strategy: given a set $\mathcal{P} = \{p_i | \forall i \in \{1, \ldots, S\}\}$ of S plans along with their plan and estimation risks (R_p and R_e respectively), and fractions of plans to be pruned with the highest plan and estimation risks (f_{pr} and f_{er} respectively), thresholds ϕ_{pr} and ϕ_{er} are computed for plan and estimation risks, respectively. Then, any plan with either plan or estimation risks

Algorithm 4.2 Conservative Plan Selection $(\mathcal{P}, \mathcal{C}, f_s)$

Inputs:
$\mathcal{P} = \{p_i \,|\, \forall i \in \{1, \ldots, S\}\}$ Set of plans to be evaluated
$\mathcal{C} = \{C_{p_i} \sim N(\mu_{p_i}, \sigma^2_{p_i}) \,|\, \forall i \in \{1, \ldots, S\}\}$ cost distributions
f_s: a value greater than 0

Output: Selected plan P^*
Initialize $P^* \leftarrow \emptyset$
Initialize $CC \leftarrow \emptyset$ an empty set for conservative cost estimates
for $p_i \in \mathcal{P}$ **do**
    $CC_i :\, = \mu_{p_i} + f_s \cdot \sigma_{p_i}$
    $CC \leftarrow CC \cup \{CC_i\}$
end for
$y^* :\, = \arg\min_{i \in \{1,\ldots,S\}} CC$
$P^* \leftarrow p_{y^*}$
return P^*

above the designated thresholds is eliminated. Parameters f_{pr} and f_{er} are tuned using the samples in the validation set. Choosing the plan from the remaining ones relies on a plan selection strategy.

4.4 Theoretical Framework Summary

This chapter established a comprehensive theoretical framework for robust query optimization based on approximate probabilistic machine learning. We systematically addressed the fundamental challenges of uncertainty in plan evaluation and selection through three key contributions. First, we formalized the notion of robustness by decomposing uncer-

Algorithm 4.3 Pruning by plan and estimation risks $(\mathcal{P}, R_e, R_p, f_{er}, f_{pr})$

Inputs:
$\mathcal{P} = \{p_i \mid \forall i \in \{1, \ldots, S\}\}$: a set of plans to be evaluated
$R_e = \{R_e(p_i) \mid \forall i \in \{1, \ldots, S\}\}$: a set of estimation risks
$R_p = \{R_p(p_i) \mid \forall i \in \{1, \ldots, S\}\}$: a set pf plan risks
f_{er}: a fraction of plans to be pruned by estimation risk
f_{pr}: a fraction of plans to be pruned by plan risk
Output: $P' \subset \mathcal{P}$: the pruned search space
$\varphi_{er} = \text{sorted}(R_e)[\lfloor |\mathcal{P}| \times f_{er} \rfloor]$
$\varphi_{pr} = \text{sorted}(R_p)[\lfloor |\mathcal{P}| \times f_{pr} \rfloor]$
$P' = \mathcal{P}$
for $i = 1$ to S **do**
    if $R_e(p_i) > \varphi_{er}$ **or** $R_p(p_i) > \varphi_{pr}$ **then**
        $P' = P' \setminus \{p_i\}$
    end if
end for
return P'

tainty into plan structure and cost model limitations, providing precise definitions for plan robustness and cost model robustness. Second, we introduced principled quantification measures for plan risk, estimation risk, and suboptimality risk, specifically tailored for ML-based cost models. Third, we presented novel risk-aware plan evaluation and selection strategies, including conservative plan selection and search space pruning algorithms that leverage these risk measures to ensure robustness in practical implementations.

The theoretical foundation established in this chapter provides the mathematical rigor necessary for developing practical robust query optimization systems. The risk quantification measures and selection strategies offer concrete mechanisms for handling uncertainty

in real-world database environments, where cost estimates are inherently imperfect and data characteristics may vary significantly.

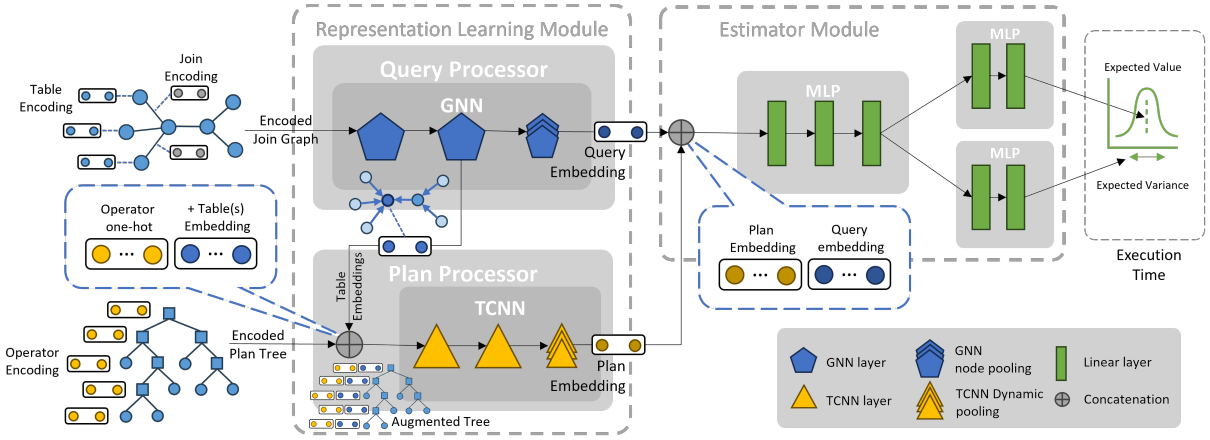
While the theoretical framework provides the mathematical foundation for robust query optimization, its practical implementation requires a sophisticated cost model capable of quantifying uncertainty and supporting risk-aware decision making. The next chapter, Chapter 5, presents the Risk-aware Learned Cost Model (RLCM) architecture that implements the theoretical concepts introduced here.

Chapter 5

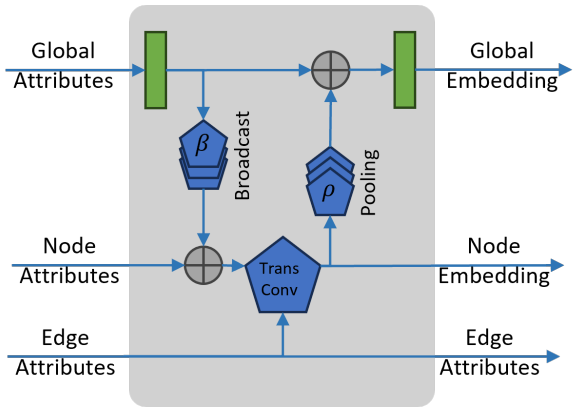
Model Architecture: Risk-aware Learned Cost Model

This chapter presents the model architecture of a risk-aware learned cost model (RLCM), implementing the theoretical framework established in Chapter 3. The RLCM represents a significant advancement over traditional cost models by capturing not only the expected execution time but also quantifying plan, estimation, and suboptimality risks through uncertainty modeling. Figure 5.1a illustrates the architecture's innovative design, which consists of three primary modules: an encoding component and transforms query-plan pairs into graph and tree vectors, a representation learning component that processes query and plan structures through specialized neural networks, and an estimation module that lever-

ages these representations to predict execution times with associated uncertainties. The chapter begins with Section 5.1 detailing the query encoding approach, which transforms join graphs into meaningful embeddings with both node and edge attributes. Section 5.2 follows with the plan encoding methodology that captures execution plan structures as vectorized trees. Section 5.3 then explains the representation learning components—the Query Processor and Plan Processor—that generate rich embeddings from these encodings. Section 5.4 describes the estimation module that combines these representations to predict both execution times and uncertainty measures. The chapter concludes in Section 5.5 by highlighting the novel aspects of the RLCM architecture compared to existing approaches. Through this design, the RLCM addresses the fundamental limitations of conventional cost models while providing the predictive capabilities necessary for risk-aware query optimization.



(a) Roq's Learned Cost Model Architecture



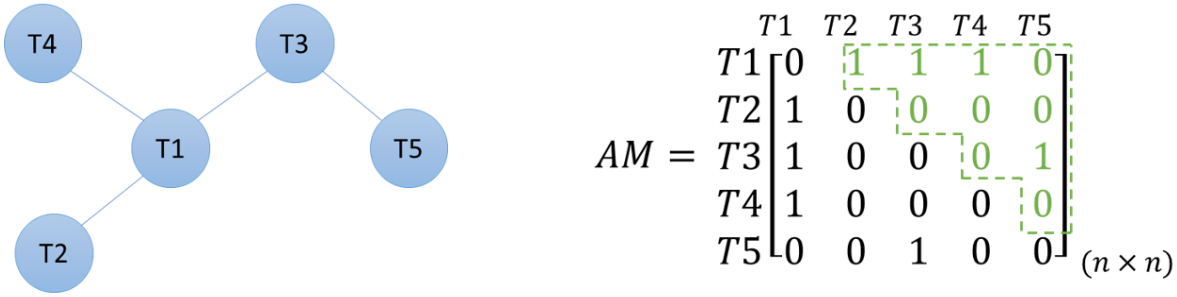
(b) The architecture of the Graph Neural Networks used in Roq

Figure 5.1: a) Architecture of the risk-aware learned cost model, including representation learning and estimator components. This architecture allows quantification of model and data uncertainties, b) Architecture of the extended transformerConv GNN model block that enables receiving and processing graph level attributes in addition to node and edge attributes

5.1 Query Encoding

For the query representation we use the query's join graph to encode information about the tables, local predicates, join predicates, aggregations, etc. Similar approaches are used by prior works [167]. In this approach we encode table statistics, such as cardinality and selectivity from local predicates, and correlations between the predicate columns (captured by the optimizer statistics) as node attributes. We also incorporate characteristics of the join predicates, such as join type, join predicate operator, join column skewness, and join selectivity, as edge attributes. The optimizer statistics are extended to capture the join characteristics too. In addition to the node and edge attributes, we capture high-level characteristics of the join graph, such as number of tables and joins, and the join graph topology, as global attributes. This representation is agnostic to the plan (join orders and plan operators) that will be used to execute the query. The following sections describe the details of this encoding approach.

5.1.1 Join Characteristics Encoded as Edge Attributes



(a) Join Graph *(b) Adjacency Matrix*

Figure 5.2: An example Join Graph and the corresponding Adjacency Matrix

The adjacency matrix of the join graph is used as a building block to encode information about the joins. For a given query that joins a set of n tables, the adjacency matrix can be defined as an $n \times n$ matrix $\mathbf{AM}_{n \times n}$ where:

$$\mathbf{AM}_{ij} = \begin{cases} 1, & \text{if table } i \text{ is joined with table } j \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

As an example, consider a query with the join graph illustrated in Figure 5.2-a, where T_i represent different tables involved. The adjacency matrix of the join graph can be defined as shown in Figure 5.2-b.

Let's revise the adjacency matrix definition to reflect that instead of a binary value (1/0), each edge has a vector of attributes associated with it if a join exists, and is a zero vector otherwise.

$$\mathbf{AM}_{ij} = \begin{cases} \mathbf{v}_{ij}, & \text{if table } i \text{ is joined with table } j \\ \mathbf{0}, & \text{otherwise} \end{cases} \tag{5.2}$$

where $\mathbf{v}_{ij}$ is a vector containing the attributes of the join edge between table i and table j, and $\mathbf{0}$ is a zero vector of the same dimension as $\mathbf{v}_{ij}$, representing the absence of a join.

$\mathbf{v}_{ij}$ can be formally defined as:

$$\mathbf{v}_{ij} = \left[a_{ij}^{(1)},\ a_{ij}^{(2)},\ \ldots,\ a_{ij}^{(d)} \right] \tag{5.3}$$

where $a_{ij}^{(k)}$ denotes the k^{th} attribute associated with the join between table i and table j, and d is the total number of attributes.

Since the join graph is undirected, its adjacency matrix is symmetric. This means that the elements on one side of the diagonal axis plus the diagonal itself carry as much information as the entire matrix.

Furthermore, note the zero elements on the matrix do not carry any information. There-

fore, the adjacency matrix $\mathbf{AM}_{n\times n}$ is a sparse representation that can be further condensed by enumerating only the elements that correspond to an edge. This is done using two matrices $\mathbf{I}_{2\times m}$ and $\mathbf{A}_{m\times d}$, where $\mathbf{I}_{2\times m}$ contains the edge indices and $\mathbf{E}$ contains the edge attributes. m represents the number of edges and d represents dimensionality of edge attributes. In graph neural network frameworks such as Pytorch Geometric [36, 120], both dense and sparse representations of the graph can be used interchangeably.

The join graph is used as a building block to encode the join attributes that are essential for a model to effectively learn its representation.

5.1.2 Edge Feature Encoding for Join Graphs

For a query joining n tables, the information about each join is encoded into a vector of multiple attributes. Each join (i.e., each edge between two tables) is associated with a feature vector $\mathbf{v}_{ij}$, capturing a set of characteristics. These vectors are ultimately concatenated to represent the complete set of join features for the query. The following describes the individual components of $\mathbf{v}_{ij}$ in detail:

Join Types

Join types (such as *inner-join*, *left outer-join*, *anti-join*) are represented using a one-hot encoded vector extended with an additional element indicating the order of the join. For a given join between tables T_i and T_j, the join type feature vector is denoted by:

$$\mathbf{v}_{ij}^{\text{type}} \in \mathbb{R}^{m+1}$$

where m is the number of supported join types, and the last element captures the ordinal position of the join in the query plan. For instance, consider the following sequence of joins:

- (T_1 inner-join T_2 ON $T_1.C1 = T_2.C1$)
- (T_3 anti-join T_4 ON $T_3.C2 = T_4.C2$)
- (T_1 left outer-join T_3 ON $T_1.C3 = T_3.C3$)

These would be encoded as:

$$\mathbf{v}_{12}^{\text{type}} = (1, 0, 0, 1),\ \mathbf{v}_{34}^{\text{type}} = (0, 0, 1, 2),\ \mathbf{v}_{13}^{\text{type}} = (0, 1, 0, 3)$$

where the first $m = 3$ elements correspond to one-hot encoded join types, and the fourth

element indicates the execution order.

Join Operators

Each join may involve one or more predicates associating columns on either side of the join using operators such as $<$, $>$, $\leq$, $\geq$, $=$, LIKE, and IN. These are encoded using a bit vector:

$$\mathbf{v}_{ij}^{\text{op}} \in \{0,1\}^k$$

where k is the number of supported operators. Each bit in the vector indicates the presence or absence of a corresponding operator in the join predicates. For example:

- A join using the $<$ operator: $\mathbf{v}_{ij}^{\text{op}} = (1,0,0,0,0,0,0)$
- A join using both $<$ and $=$ operators: $\mathbf{v}_{ij}^{\text{op}} = (1,0,0,0,1,0,0)$

Ratio of Join Column(s) Cardinality to Table Cardinality

The ratio of the number of distinct values in the join columns to the table cardinality provides a signal about column uniqueness. This value, between 0 and 1, is computed using available table and column statistics, including *COLCARD* and *COMBINED_COLCARD*. When multiple columns are involved in the join on one side, *COMBINED_COLCARD* is

used. Otherwise, the individual column cardinality applies. The feature vector is:

$$\mathbf{v}_{ij}^{\text{col\_ratio}} \in \mathbb{R}^2$$

containing one value for each side of the join.

Ratio of Table Cardinalities

The relative sizes of the two joined tables also influence the join result size. This ratio is computed at compile time from optimizer statistics:

$$\mathbf{v}_{ij}^{\text{tbl\_ratio}} \in \mathbb{R}$$

representing the ratio $\text{Card}(T_i)/\text{Card}(T_j)$.

Skewness of the Join Columns

Post-predicate, the distribution of values in join columns may show skewness, affecting cardinality. Skewness is captured via a metric such as the *Gini coefficient*, ranging from 0 (uniform distribution) to 1 (full inequality). The skewness of the join columns on each side is encoded as:

$$\mathbf{v}_{ij}^{\text{skew}} \in \mathbb{R}^2$$

Join Factor

The Join Factor reflects the selectivity behavior of joins and is computed by:

1. Calculating the join result from sampled data (C_1)
2. Computing the cartesian product size from the sampled cardinalities (C_2)
3. Determining the ratio C_1/C_2

For joins with multiple predicates, Join Factors are captured per predicate, then the minimum, average, and maximum are taken. Thus:

$$\mathbf{v}_{ij}^{\text{jf}} \in \mathbb{R}^3$$

Inclusion Factor

This factor measures the overlap between the sets of values on both sides of a join. It is computed as:

1. Counting the distinct values from either side of the join.
2. Determining the number of matching values between the two sets.
3. Calculating the size of their cartesian product.

4. Taking the ratio of matching values to the cartesian product size.

As with the Join Factor, when multiple predicates exist, minimum, average, and maximum Inclusion Factors are computed:

$$\mathbf{v}_{ij}^{\text{if}} \in \mathbb{R}^3$$

Final Edge Feature Vector

Finally, for each join (edge) (i, j), the feature vector is formed by concatenating all these components:

$$\mathbf{v}_{ij} = \left[\mathbf{v}_{ij}^{\text{type}}, \mathbf{v}_{ij}^{\text{op}}, \mathbf{v}_{ij}^{\text{col\_ratio}}, \mathbf{v}_{ij}^{\text{tbl\_ratio}}, \mathbf{v}_{ij}^{\text{skew}}, \mathbf{v}_{ij}^{\text{jf}}, \mathbf{v}_{ij}^{\text{if}}\right]$$

This structured encoding allows the join graph to capture rich semantic and statistical information for each join operation, enabling learning-based query performance prediction models to reason over the complex interdependencies of query plans.

5.1.3 Table Characteristics Encoded as Node Attributes

In addition to encoding features associated with joins (edges), several important characteristics intrinsic to the base tables are encoded as node features in the join graph. Each table T_i participating in a query is associated with a feature vector $\mathbf{u}_i$ capturing its relevant properties. These node-level features contribute essential information for learning

the representations of the join queries. The node features fall into the following three categories:

Base Table Cardinalities and Local Predicate Selectivities

The first category includes the cardinality of each base table as well as the selectivity of any local predicates applied to that table. These characteristics are encoded as part of the node feature vector:

$$\mathbf{u}_i^{\text{card-sel}} \in \mathbb{R}^2$$

where:

- The first element represents the **base table cardinality** for T_i, which is typically obtained from system catalog statistics available in the database management system (DBMS).
- The second element captures the **selectivity of local predicates** applied to T_i. This selectivity can be estimated by applying the local predicates to a sample of the table's rows or by using the optimizer's estimation techniques, such as histograms or selectivity models.

These features are crucial because they directly influence the estimated size of the intermediate and final results in a join query plan.

Local Predicate Correlations

When multiple local predicates are applied to a single base table, the degree of correlation between the predicate columns affects the composite selectivity. If the optimizer estimates the composite selectivity through a model, such correlations are implicitly handled. However, in cases where precise estimates are unavailable, it becomes necessary to explicitly capture the pairwise correlations between the columns involved in the local predicates.

These correlation factors are computed offline and stored within the database's extended statistics. For a table T_i with multiple local predicates, all pairwise correlations are retrieved from these stored statistics, and their:

- minimum,
- average, and
- maximum

values are computed to summarize the correlation structure. These three values are then appended as additional elements in the node feature vector:

$$\mathbf{u}_i^{\text{corr}} \in \mathbb{R}^3$$

Final Node Feature Vector

The complete node feature vector for each table T_i is constructed by concatenating the features described above:

$$\mathbf{u}_i = \left[\mathbf{u}_i^{\text{card-sel}},\ \mathbf{u}_i^{\text{corr}}\right]$$

This final vector incorporates table cardinality, local predicate selectivity, and correlations between local predicates, ensuring that the join graph representation comprehensively captures all relevant table-level information for query cardinality estimation.

5.2 Plan Encoding

In addition to encoding query- and join-graph level information, we capture plan-level details by encoding the structure and properties of the query execution plan as a vectorized tree, similar to the method proposed by Neo [101]. Each node in this tree corresponds to an operator node in the query plan and is represented by a feature vector $\mathbf{p}_k$.

For a given query execution plan, the plan-level encoding consists of a tree structure, where each node's feature vector captures the following components:

- **Operator Type Encoding:** A one-hot vector indicating the operator type (e.g.,

hash join, *merge join*, *loop join*, *aggregate*, *index scan*, *table scan*). Denoted by:

$$\mathbf{p}_k^{\text{op}} \in \{0,1\}^{|J|}$$

where $|J|$ is the total number of supported operator types.

- **Table Participation Encoding:** An encoding that tracks the set of base tables involved in the subgraph rooted at the current plan node. Each table T_i is represented by a one-hot vector of length $2|R|$, where $|R|$ is the number of base relations, and each table has two bits: one for indicating a *table scan* and one for an *index scan*. Specifically:

$$\mathbf{p}_k^{\text{tbl}} \in \{0,1\}^{2|R|}$$

For internal (join) nodes, these entries are the union of the corresponding child nodes' participation vectors. For leaf nodes, a one-hot vector indicates the scan type(s) for the base table involved. If a scan is unspecified, both positions (table and index scan) are set to 1.

- **Subtree Join Graph Embedding:** To capture the structural and statistical information associated with the join subgraph rooted at a given plan node, we compute a learned embedding by processing the join graph encoding corresponding to the

involved tables and joins through a stack of graph neural networks, referred to as the *Query Processor*. The Query Processor operates on the previously defined node feature vectors $\mathbf{u}_i$ and edge feature vectors $\mathbf{v}_{ij}$ associated with the join graph.

Formally, let $G_k = (V_k, E_k)$ denote the subgraph of the overall query join graph corresponding to the subtree rooted at plan node k, with:

- $V_k \subseteq \{T_1, \ldots, T_n\}$ as the set of involved base tables.
- E_k as the set of joins (edges) among these tables.

The details of the Query Processor are described in section 5.3.1

The subtree join graph embedding for plan node k is formally defined as:

$$\mathbf{p}_k^{\text{jg}} = \text{QueryProcessor}(G_k)$$

where QueryProcessor(G_k) applies the Query Processor (described in Section 5.3.1) to the subgraph G_k to produce a learned embedding that captures the structural and statistical properties of the join subgraph rooted at plan node k.

Final Plan Node Vector

The complete feature vector for each plan tree node is constructed by concatenating these components:

$$\mathbf{p}_k = \left[\mathbf{p}_k^{\text{op}},\ \mathbf{p}_k^{\text{tbl}},\ \mathbf{p}_k^{\text{jg}}\right]$$

Each query execution plan is thus encoded as a tree of such vectors, preserving both the hierarchical plan structure and the rich relational semantics captured via the learned join graph representations. This encoding enables the use of tree-based neural models for plan-level tasks such as cost prediction or operator selection.

5.3 Representation Learning

The representation learning module takes both the query-level and plan-level encodings as input and produces two types of embeddings:

- A **query embedding** $\mathbf{e}^{\text{query}}$ via the *Query Processor*.
- A **plan embedding** $\mathbf{e}^{\text{plan}}$ via the *Plan Processor*.

While we partially adopt the Plan Processor architecture from prior works [101], our

design of the Query Processor and its integration with the Plan Processor is novel and contributes to the robustness and accuracy of our model, as demonstrated in Section 6.

5.3.1 Query Processor

The Query Processor operates on the encoded join graph $G = (V, E)$, where each node (table) $T_i \in V$ is associated with a feature vector $\mathbf{u}_i$ and each edge (join) $(i, j) \in E$ with a feature vector $\mathbf{v}_{ij}$. We use a stack of L Graph Neural Network (GNN) layers for iterative message passing over the graph, following:

$$\mathbf{h}_i^{(l)} = \phi^{(l)} \left(\mathbf{h}_i^{(l-1)}, \left\{ \psi^{(l)} \left(\mathbf{h}_i^{(l-1)}, \mathbf{h}_j^{(l-1)}, \mathbf{v}_{ij}, \mathbf{g} \right) \mid j \in \mathcal{N}(i) \right\} \right) \tag{5.4}$$

where:

- $\mathbf{h}_i^{(l)}$ is the representation of node T_i at GNN layer l, initialized as $\mathbf{h}_i^{(0)} = \mathbf{u}_i$.
- $\mathcal{N}(i)$ denotes the neighbors of node i in the join graph.
- $\mathbf{v}_{ij}$ is the edge feature vector between T_i and T_j.
- $\mathbf{g}$ represents graph-level attributes broadcast to all message computations.
- $\phi^{(l)}$ is the node update function (TransformerConv block).

- $\psi^{(l)}$ is the attention-based message function.

We implement $\phi^{(l)}$ using TransformerConv [134], an attention-based GNN operator extending the standard message passing framework by conditioning messages not only on the source and destination nodes, but also on the edge features and global graph attributes (see Figure 5.1b). This design is well-suited for join graphs where the influence of neighboring tables depends on both table properties and join semantics.

After L layers of message passing, we obtain the final table embeddings $\mathbf{h}_i^{(L)}$ for all $T_i \in V$. To obtain a fixed-size embedding for the entire query (join graph), we apply mean and max pooling over the final node embeddings:

$$\mathbf{e}^{\text{query}} = \left[\text{MEAN}\left(\left\{\mathbf{h}_i^{(L)} \mid T_i \in V\right\}\right),\ \text{MAX}\left(\left\{\mathbf{h}_i^{(L)} \mid T_i \in V\right\}\right)\right] \tag{5.5}$$

This two-pooling strategy ensures that both the average behavior and dominant patterns in the join graph are captured. Figure 5.1b illustrates the architecture of our extended TransformerConv-based Query Processor.

Formally, we define the Query Processor as a function that takes a join graph G and returns its embedding:

$$\text{QueryProcessor}(G) = \mathbf{e}^{\text{query}} \tag{5.6}$$

where $\mathbf{e}^{\text{query}}$ is computed as described above.

5.3.2 Plan Processor

The Plan Processor operates on the vectorized query plan tree, where each plan node k is associated with a feature vector $\mathbf{p}_k$ as defined in Section 5.2. To enrich the plan encoding with semantic information from the join graph, we augment each plan node with relevant table embeddings obtained from the Query Processor.

Specifically:

- If plan node k is a **leaf access operator** (e.g., table scan or index scan), it is augmented with the embedding of the corresponding table T_i:

$$\mathbf{a}_k = \mathbf{h}_i^{(L)}$$

- If plan node k represents a **subgraph operator** (e.g., join) involving multiple tables $T_{i_1}, T_{i_2}, \ldots, T_{i_s}$, their embeddings are aggregated via mean pooling:

$$\mathbf{a}_k = \text{MEAN}\left(\left\{\mathbf{h}_i^{(L)} \mid T_i \in V_k\right\}\right)$$

where $V_k \subseteq V$ is the set of tables participating in the subtree rooted at plan node k.

The augmented plan node feature is then:

$$\mathbf{p}'_k = [\mathbf{p}_k,\ \mathbf{a}_k]$$

The augmented plan tree is processed through a stack of M Tree Convolutional Neural Network (TCNN) layers [109], which propagate information from child to parent nodes along the tree hierarchy. After M layers, dynamic pooling is applied to produce a fixed-size embedding representing the entire query plan:

$$\mathbf{e}^{\text{plan}} = \text{POOL}\left(\left\{\mathbf{h}_k^{(M)} \mid \text{plan node } k\right\}\right) \tag{5.7}$$

This architecture is effective for processing hierarchical plan structures, as demonstrated by prior works [100, 101, 167].

Formally, we define the Plan Processor as a function that takes a vectorized plan tree P and returns its embedding:

$$\text{PlanProcessor}(P) = \mathbf{e}^{\text{plan}} \tag{5.8}$$

where $\mathbf{e}^{\text{plan}}$ is computed as described above.

5.4 Estimation Module

The Estimation Module combines the learned query and plan embeddings to predict the expected execution time and its associated uncertainty. First, the two embeddings are concatenated:

$$\mathbf{e} = \left[\mathbf{e}^{\text{query}},\ \mathbf{e}^{\text{plan}}\right] \tag{5.9}$$

This combined representation is then processed through a multi-layer perceptron (MLP) $\mathcal{F}(\cdot)$ to learn their joint association:

$$\mathbf{z} = \mathcal{F}\left(\mathbf{e}\right) \tag{5.10}$$

Finally, two separate MLP branches are applied to predict the execution time and its variance:

$$\hat{t} = \mathcal{G}_1\left(\mathbf{z}\right) \tag{5.11}$$

$$\hat{\sigma}^2 = \mathcal{G}_2\left(\mathbf{z}\right) \tag{5.12}$$

where:

- $\hat{t}$ is the estimated execution time.
- $\hat{\sigma}^2$ is the estimated variance (uncertainty) of the execution time prediction.

5.5 Novelty of Roq's Model Architecture

While certain components of Roq's model architecture are inspired by prior works, its overall composition and extensions are novel. In particular:

- The use of an extended TransformerConv GNN to learn robust, generalizable table and query representations from join graphs.
- The seamless integration of these learned query embeddings into TCNN-based plan processors.
- The dual-branch estimation module that simultaneously predicts query execution time and its associated risk (variance).

This combination of advanced graph representation learning and plan-tree processing for cost estimation and uncertainty modeling is, to our knowledge, novel within the literature.

Chapter 6

Experimental Study

This chapter presents a comprehensive experimental study evaluating the effectiveness of the robust query optimization approach proposed in this thesis. Through rigorous empirical analysis, it validates the theoretical framework and model architecture introduced in previous chapters while demonstrating significant performance improvements over state-of-the-art alternatives. Section 6.1 outlines the experimental setup, detailing the hardware and software environment, datasets and workloads used, model training methodology, implemented baselines, and evaluation metrics. Section 6.2 then presents the experimental results across five key dimensions: prediction accuracy compared to baseline approaches, the impact of the proposed model architecture on plan quality, the effectiveness of various risk-aware plan selection strategies, the influence of different types of uncertainties,

and robustness to workload shifts. This section also examines the computational overhead of the risk quantification process, establishing its practicality for real-world deployment. Throughout the chapter, the results consistently demonstrate that accounting for uncertainties in cost estimation leads to more robust query execution plans with significantly reduced worst-case performance degradation, while maintaining competitive average performance. These findings provide compelling evidence for the efficacy of uncertainty-aware cost modeling in addressing the fundamental challenges of robust query optimization.

6.1 Experimental Setup

This section provides a comprehensive overview of the experimental setup used to evaluate Roq's performance against state-of-the-art approaches. It begins by detailing the hardware and software environment in which the experiments were conducted, including the database management system and development tools used. The section then describes the diverse datasets and workloads employed for testing, encompassing both real-world and synthetic benchmarks with varying characteristics and complexity levels. Next, it outlines the model training methodology and parameter tuning process that ensures optimal performance. The section also introduces the baseline systems against which Roq is compared, including both traditional query optimizers and advanced learning-based ap-

proaches. Finally, it presents the evaluation metrics used to measure performance across multiple dimensions, including prediction accuracy, plan quality, and runtime efficiency. This thorough experimental framework enables a rigorous assessment of Roq's capabilities in robust query optimization.

6.1.1 Database and Environment Settings

The experiments are performed on a Windows machine with a 24-core Core i7 CPU, 64 GB of memory, and a 16 GB NVIDIA GeForce RTX 4080 GPU (used only for model training). Db2 v11.5 is used as the RDBMS for compiling and running the queries. Plans obtained from the Db2 optimizer as well as the best-performing plans are used as baselines for evaluating the plans selected by the proposed approaches. We consider the plan with the minimum execution time among all plans in the search space as the best-performing plan. The prototype code [1] is written in Python 3.8, with the deep learning components written using Pytorch Geometric [36, 120]. Parameter tuning is performed using Ray Tune [1].

6.1.2 Datasets and workloads

We use the following benchmarks to address the specific requirements of different database stakeholders: The Join Order Benchmark (JOB) [88], the Cardinality Estimation Bench-

[1] The code is made available at: https://github.com/M2oDA-Lab/Roq

mark (CEB) [114], and the Decision Support Benchmark (DSB) [112].

JOB is based on the real-world Internet Movie Database (IMDB), representing everyday web application scenarios that directly impact end-user experience. This workload helps us evaluate whether our approach can deliver the consistent performance and minimal regressions that end-users require for responsive applications.

CEB, also based on IMDB but with entirely different query templates, allows us to test Roq's robustness against workload shifts—a critical capability for database administrators who need systems that adapt to changing query patterns without manual intervention.

DSB leverages TPC-DS, a realistic synthetic dimensionally modeled dataset typically used in enterprise analytics. This benchmark represents OLAP workloads commonly found in business intelligence applications, helping us assess whether our approach can provide the performance stability needed by database administrators managing mission-critical reporting systems.

Additionally, we developed TPC-DS+, a randomly generated workload based on TPC-DS with more challenging scenarios and ad-hoc query patterns. This benchmark simulates unpredictable environments where queries cannot be anticipated in advance, directly addressing database developers' requirements for solutions that automatically adapt to new patterns without manual reconfiguration or tuning.

The characteristics of these workloads are summarized in Table 6.1. Training data includes pairs of queries and plans based on these benchmarks, encoded to be consumable by an ML model, as described in the following sections.

CEB 1000 queries are randomly sampled from various query templates in the CEB with up to 12 joins over 21 tables from the IMDB dataset. These are split to non-overlapping subsets of 800, 100, and 100 queries for training, validation, and test sets. Each experiment is repeated 5 times with different random seeds and the average results are reported.

JOB contains 113 queries with up to 27 joins over 21 tables from the IMDB dataset. Given the small sample size, 10-folds cross validation is used to train models using this dataset. In each fold, 80% are used for training, 10% for validation, and 10% for testing, such that over the 10 folds each query appears in the test set exactly once. The aggregated test results from different folds are reported.

TPC-DS+ dataset is used as the third benchmark. The original benchmark includes only referential integrity (RI) joins, hence star-schema and snowflake-schema queries only. It does not include many-to-many joins or cyclic joins. In order to enforce more challenging scenarios, we extend this benchmark by adding a) many-to-many joins, b) cyclic joins, c) local predicate on new columns that are correlated with join predicate columns. This extended dataset is used to randomly generate 13k synthetic queries with various query templates and join patterns. The characteristics of these queries are controlled by

parameters that a query generator takes as input, such as the number of joins, join types (inner-join, outer-join, left/right outer join, anti-join), number of join and local predicates, join and local predicate operator types ($==, <, >, <=, >=$), etc. Each query has up to 4 joins with each join having up to 3 join predicates and each table with up to 5 local predicates. The generated queries are uniformly distributed across different numbers of joins, such that 1-join queries are as frequent as 2-, 3-, and 4-join queries. The join predicates are uniformly sampled from the referential integrity joins. The queries may exhibit various join graphs (linear, star, snowflake, cyclic). The predicates used for each join as well as the local predicates defined on each base table may have correlations to simulate more realistic scenarios. The queries may also exhibit join-crossing correlations between the predicate columns across different joins and local predicates. As such, this workload ensures sufficient randomness and complexity in the query patterns to represent more challenging scenarios for generalization. 12k of the queries are used for training, 500 for validation, and 500 for testing. Each experiment is repeated five times with different random seeds and the average results are reported. This benchmark is used for an ablation study that evaluates the impacts of accounting for different types of risk.

DSB 1000 queries are generated based on the Decision Support Benchmark (DSB) [29], an extension of TPC-DS that enhances data generation by introducing more realistic, skewed, and correlated data distributions rather than the mostly independent, uniform

Table 6.1: Benchmarks used in the experiments

Workload	Queries	Templates	Max Joins	Dataset	Tables	Type
CEB	1000	16	12	IMDB	21	Real
JOB	113	33	28	IMDB	21	Real
TPC-DS+	13000	N/A	4	TPC-DS	24	Synthetic
DSB	1000	15	13	TPC-DS	24	Synthetic

distributions in TPC-DS. It augments query templates with predicate filters for fine-grained data slicing and adds new templates featuring more complex join patterns (such as non-equi and many-to-many joins), resulting in many more distinct query instances. The queries are generated based on 15 distinct query templates and are split into 800, 100, and 100 queries for training, validation, and test sets, respectively. Each experiment is repeated 5 times with different random seeds and the average results are reported.

6.1.3 Training Samples Generation

Plan Generation Each query is compiled with various hint sets and the different plans generated by the optimizer are captured. The first compilation is done without any constraints, giving the default plan chosen by the optimizer. Additionally, each query is compiled using 13 hint sets (see Appendix A). Each hint set imposes a set of constraints on join and access operators to be used in the query plan. Alternatively, randomly generated plans could be used. However, fully random plans can be arbitrarily suboptimal and of low quality. Hint sets produce locally optimum plans that are expected to be better

than the optimizer's plan in some scenarios, as shown by Bao [100]. While this does not represent the way the optimizer traverses the search space, it is sufficient for providing a diversified set of potentially optimal plans for the purpose of this study.

Collecting Labels Each query is executed using each of the guidelines and the execution time is measured. A timeout threshold is used to terminate too long executions. This threshold is dynamically set to be 10 times the time it takes to run the query using the best performing plan found so far, rounded up to the nearest greater integer. This threshold allows a higher level of exploration to obtain information on more risky plans.

Preprocessing Preprocessing is performed for all splits based on the statistics obtained from the training data. The input data is preprocessed by performing null imputation and min-max scaling. Null values are replaced by the mean value of the respective features. Then min-max scaling brings the range of the values for each feature between zero and one: $X^t = X_{scaled} = \frac{X - \min(X)}{\max(X) - \min(X)}$. The labels are pre-processed by applying logarithmic transformation and min-max scaling. The execution times can range greatly and typically exhibit a large skewness towards zero. Log transformation is applied to the labels to reduce skewness. This is performed with the following formula, where y_{log} represents the transformed label: $y_{log} = \log_{10}(y)$. The Min-Max scaling then brings the range of values between zero and one. This is a desirable range for deep learning models that use the Sigmoid activation function in their output layers: $y^t = y_{scaled} = \frac{y_{log} - \min(y_{log})}{\max(y_{log}) - \min(y_{log})}$.

6.1.4 Model Training and Parameter Tuning

To avoid over-fitting, three mechanisms are used in model training: early stopping, dropout, and reducing learning rate on plateau. Applying dropout at every layer is additionally necessary to enable the variational inference needed to obtain model uncertainty. The dropout rate along with other architectural parameters such as the number of layers, number of neurons, and learning rate are tuned using the Asynchronous Successive Halving Algorithm [91]. This algorithm allows for exploring a large number of parameter combinations while limiting the total amount of time needed for tuning.

6.1.5 Baselines

We compare Roq against a classical RDBMS optimizer, that of IBM Db2, as well as four prominent works, Neo [101], Bao [100], Lero [179], and Balsa [162]. Neo and Bao are selected due to their demonstrated robustness to errors in input features. Balsa is selected due its to demonstrated robustness to workload shifts. Lero is selected due to its demonstrated minimization of regressions. Each approach is evaluated for its ability to select a robust plan from the same set of plans for a given query. To this end, the search space is identical for all approaches and is enumerated as explained in the previous section. The plans selected by each approach are executed using Db2's execution engine.

IBM Db2 serves as a baseline representing the state-of-art classical query optimizers in our experiments. In our tests, Db2's optimizer captures Column Group Statistics (CGS) to account for correlations between database attributes. This feature reduces the impact of the independence assumption which leads to severe cardinality and cost underestimations. The performance and robustness of the plans selected by Db2 serve as a baseline.

Neo is a fully learned query optimizer that uses deep neural networks and reinforcement learning to generate efficient query execution plans. It starts by bootstrapping from traditional optimizers and then continuously refines its decision-making by learning from actual query execution times [2].

Bao uses a more lightweight cost model that takes plans as input and predicts latency. At each node of the vectorized plan tree, in addition to node type, it captures the estimated cardinality and cost.

Balsa is a learned query optimizer that avoids relying on expert demonstrations. It bootstraps from a simple simulator and then fine-tunes its deep reinforcement learning model using real query execution latency feedback.

Lero is a more recent work that proposes a learning-to-rank approach to a learned query optimizer. It learns to compare pairs of plans in the search space and create a

[2]Our implementation of Neo uses one-hot encoding for table representation. In order to provide it with the knowledge of the underlying data, we add cardinality estimates to encodings for each plan node.

total order. The model is trained using a pairwise ranking loss function. Additionally, it enumerates pairs of subplans traversed during search and uses them as additional samples to train the model.

6.1.6 Evaluation Measures

We design five sets of experiments to evaluate Roq based on prediction error, correlations with runtime, robustness to workload shifts, the inference overheads, and an ablation study on the role of plan and estimation risks. Our comparisons employ the following measures.

Prediction Error. The predictive performance of the model is evaluated using various measures depending on the subject under consideration. The accuracy of the predicted execution times is evaluated by the Q-error [108]. Given the actual label y and the estimated value $\hat{y}$, Q-error is defined as: $Q_{error} = \frac{\max(y,\hat{y})}{\min(y,\hat{y})}$.

Correlation. We measure the correlations between the estimates (or the optimizer's Cost) and the actual execution times using Spearman's rank correlation coefficient rather than the more widely used Pearson's correlation coefficient. While Pearson's coefficient measures the linear relationships between two variables, Spearman's coefficient measures the monotonic relationships, whether linear or not. Therefore, given the estimated costs or execution times are used to rank the plans, Spearman's coefficient is a more suitable measure of correlation in this context.

Plan Suboptimality. The quality of plans generated by a strategy is evaluated using the Suboptimality measure. The suboptimality of a plan p_i is computed with respect to the best performing plan $p*$: $\text{Subopt}(p_i) = \frac{ET(p_i)}{ET(p^*)} \in \mathbb{R}; 1 \leq \text{Subopt}(p_i) < \infty$, where $ET(p_i)$ is the execution time of the plan p_i, and $p* = argmin_{i \in \{1,\ldots,S\}} ET(p_i)$. We assume that the best performing plan among the ones generated by the hint sets is p^*. The distribution of this measure over all plans selected by a certain method, particularly the tail-end of the distribution, characterizes the worst-case performance degradation of that method compared with an oracle which always selects the best plan. This is suggested as a measure of end-to-end robustness [52].

Runtime. Improving suboptimality as a measure of robustness may lead to picking plans that are more robust but more expensive than the optimal plan. While the main goal of our work is to improve robustness, we need to demonstrate that robust query optimization does not cause a major slowdown for a given workload. Therefore, the total runtime of the workload is measured for each approach.

Inference Overheads. Using the risk-aware plan evaluation algorithms has an overhead for the inference time since multiple predictions must be made using the base model. This overhead is most significant for plan selection by suboptimality risk given the additional computations needed. Therefore, we measure the inference time for this algorithm for various inference iterations while evaluating the impact on plan suboptimality and

runtime.

6.2 Experimental Results

This section presents a detailed analysis of the experimental results that demonstrate Roq's effectiveness in robust query optimization. It begins by examining Roq's predictive performance compared to baseline approaches, showcasing superior accuracy and correlation with actual execution times. The section then evaluates the impact of Roq's innovative model architecture on plan selection quality, highlighting improvements in both robustness and runtime efficiency. Next, it analyzes the effectiveness of various risk-aware plan selection strategies, demonstrating their ability to reduce performance variability and minimize suboptimality. The section also explores the differences in performance across benchmarks, providing insights into the conditions under which Roq's benefits are most pronounced. Furthermore, it presents an ablation study that isolates the contributions of plan risk and estimation risk to overall performance. The section then demonstrates Roq's exceptional robustness to workload shifts, a critical capability for real-world database systems. Finally, it examines the computational overhead associated with Roq's inference process, confirming its practicality for deployment in production environments.

6.2.1 Prediction Accuracy

The average predictive performance of Roq's base model (without uncertainty quantification) consistently outperforms the baselines. Roq's prediction's q-error is superior to the baselines in all tested scenarios based on CEB, JOB, and DSB, as shown in Figure 6.1a [3]. The correlation of the predicted runtime with the actual runtime is more significant for Roq compared with either the baseline ML-based approaches or the Db2 optimizer's cost (denoted as *Cost*), as shown in Figure 6.1b. In these evaluations we only use the raw predictions from each model and do not use MC dropout variational inference to obtain a more accurate prediction.

[3]Note that Lero is not included in this evaluation as it is a learning-to-rank model, and that, similar to the optimizer's cost, the absolute values of its predictions do not correspond to latency. Therefore, its q-error is not meaningful.

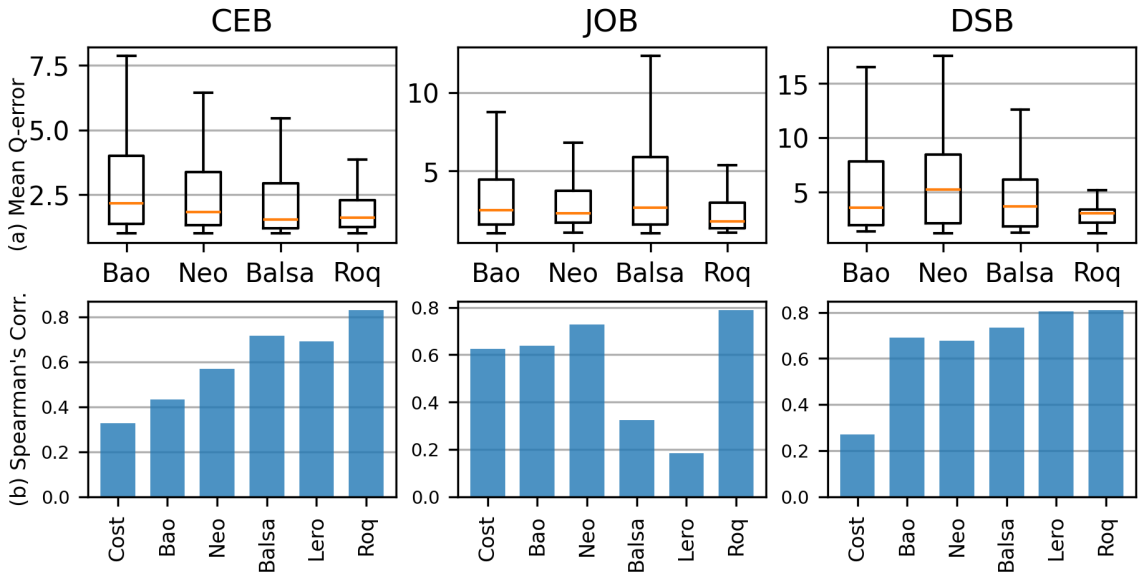


Figure 6.1: Roq's predictive performance vs. the baselines. (a) q-error, and (b) Spearman's correlation with respect to latency. The *Cost* baseline refers to the optimizer's cost estimate

6.2.2 Model Architecture

Better average predictive accuracy may not lead to better plan selection performance. Therefore, we perform experiments to evaluate the quality of the generated plans based on their suboptimality and runtime. The plans selected by the risk-aware strategies using Roq demonstrate greater robustness based on plan suboptimality and increased runtime improvements compared with the baselines, as shown in Figures 6.2 a and 6.2b respectively.

To isolate the impact of the plan selection strategies as implemented by Algorithms 4.2, 4.1, and 4.3, Roq is tested once without risk quantification, similar to the baselines.

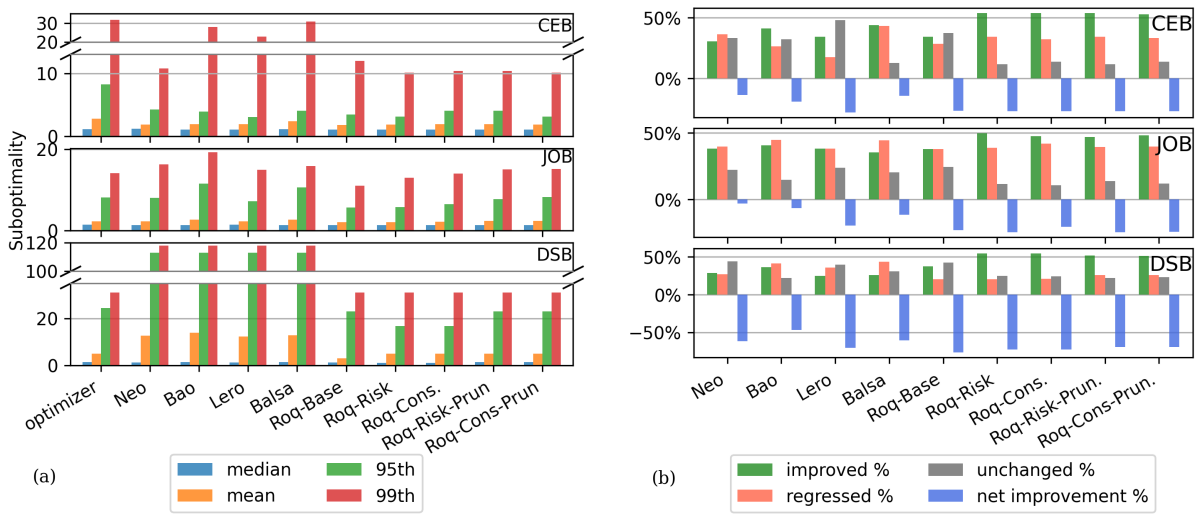


Figure 6.2: The performance of plans selected by each approach illustrated by (a) suboptimality distribution and (b) improved, regressed, and unchanged percentage of queries, as well as total runtime improvement percentage compared with Db2's plan

This scenario is denoted as the *Roq-Base*. On average, Roq's base model improves the mean and the 99^{th} percentile of suboptimality compared with the optimizer by 26.9% and 28.0%, respectively. In addition, it improves the the same measures compared with the ML-based baselines on average by 30.8.3% and 44.9.8% respectively, and it improves runtime compared with the ML-based baselines on average by 22.5%. These improvements are attributed to Roq's architecture using GNNs for producing node embeddings from the join graph, which enables enhanced generalization to the unseen test queries.

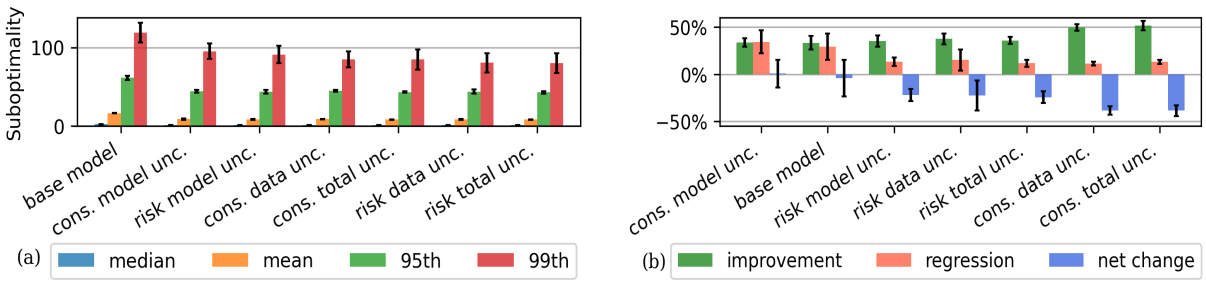


Figure 6.3: Comparing the impacts of the risk-based plan selection strategies using model, data, and total uncertainties in (a) suboptimaility distribution and (b) runtime improvements, regressions, and net change (all in percentage) for TPC-DS+ queries

6.2.3 Plan Selection Strategies

The risk-aware plan selection strategies effectively increase the number of improved queries and improve the overall performance and robustness. As demonstrated in Figure 6.2, in comparison with the ML-based baselines, plan selection by subopt risk (denoted by *Roq-Risk*) yields the greatest level of robustness by improving the mean and the 99^{th} percentile of suboptimality on average by 24.5% and 44.4% respectively. Conservative plan selection (denoted by *Roq-Cons*) improves the same measures by 22.5% and 44.4% respectively. These two strategies improve runtime compared with the ML-based baseline on average by 23.0% and 17.3% respectively. *Roq-Risk* and *Roq-Cons* improve the 99^{th} percentile of suboptimality compared with *Roq-Base* by 19.6% and 19.4%, respectively. While both *Roq-Risk* and *Roq-Cons* improved the tail-end performance similarly, the former was more effective at improving the overall runtime. These two strategies are combined with plan

pruning based on plan and estimation risks, (denoted by *Risk-Roq-Prun* and *Risk-Cons-Prun*). Pruning did not have a significant impact on the performance of either the *Roq-Risk* or *Roq-Cons.* over the three benchmarks. This is expected as these strategies adjust the predictions according to their uncertainties and effectively eliminate risky predictions by default.

6.2.4 Explaining Differences Between Benchmarks

While the above observations hold on average, the benefits of Roq and the risk-aware strategies are not the same for all three benchmarks. While they consistently improved robustness, as measured by the tail-end of suboptimality, their ultimate benefits for runtime were not the same. In general, we make this observation that Roq's benefits can be most significant when high levels of uncertainty exist. Since JOB, CEB, and DSB workloads are based on a predefined limited number of templates, they have a lower level of data uncertainty. Consequently, a model trained on such workloads is expected to behave more consistently, which means model uncertainty will be limited too. Our extended TPC-DS+ workload, on the other hand, is not generated based on predefined templates and contains queries with arbitrary join graphs and predicates. In addition, it contains more challenging scenarios such as multi-predicate joins and join-crossing correlations. These characteristics lead to a high level of data uncertainty as it becomes more difficult for the encoding to

carry relevant information about queries and plans. Model uncertainty will be high too as the model will have a more difficult job of distinguishing between various samples. This explains why the benefits of Roq's risk-aware strategies are more obvious for the TPC-DS+ workload.

6.2.5 Plan vs. Estimation Risks

To evaluate the roles of the risk of the plan (quantified by uncertainty in the data) and the estimation risks (quantified by uncertainty in the model) in the selection of the plan, we performed an ablation study in which risk-aware plan selection strategies are tested using model, data, or total uncertainties. We use the extended TPC-DS+ benchmark for which we observed a more significant impact from risk-aware plan selection methods. The quality of the resulting plans is evaluated in terms of run-time, improvements or regressions, as well as the suboptimality of the resulting plans (Figures 6.3a and 6.3b). Using either model or data uncertainty to quantify risks in each plan selection strategy provides significant improvements compared to the base model, and using total uncertainties provides maximum improvements. This suggests that the plan and the estimation risks each play a role in improving Roq's performance and the impact is maximized if they are used together. In addition, the impact of accounting for plan risk is more significant for achieving better runtime performance since it effectively avoids long-running plans and thus significantly

reduces regressions.

To complement these results, we run similar experiments using JOB, CEB, and DSB too. The suboptimality results are summarized in Table 6.2. The results confirm that the above observations hold even for workloads based on query templates (i.e. with lower uncertainty). In these cases, too, accounting for data uncertainty has a more significant impact on performance improvement compared to accounting for model uncertainty. Furthermore, the highest level of improvements are observed when using the total uncertainty.

Table 6.2: Comparing the impacts of the risk-based plan selection strategies using model, data, and total uncertainties in suboptimaility distribution for CEB, JOB, and DSB queries.

	CEB					JOB					DSB				
Suboptimality	**median**	**mean**	**95th**	**99th**	**max**	**median**	**mean**	**95th**	**99th**	**max**	**median**	**mean**	**95th**	**99th**	**max**
base model	**1.05**	1.16	1.44	2.68	3.51	1.18	1.26	1.86	2.87	3.51	6.33	5.85	13.35	16.58	20.52
cons. model unc.	1.14	1.16	1.59	1.85	**1.99**	**1.17**	1.20	1.72	1.94	**1.98**	5.49	6.02	12.87	14.75	**19.76**
cons. data unc.	1.10	**1.13**	**1.27**	**1.80**	2.00	**1.17**	**1.18**	**1.56**	**1.83**	**1.98**	8.44	7.05	13.94	17.10	27.21
cons. total unc.	1.10	**1.13**	**1.27**	**1.80**	2.00	**1.17**	**1.18**	**1.56**	1.84	**1.98**	4.60	**5.10**	11.96	14.96	28.71
risk model unc.	1.14	1.16	1.47	1.85	**1.99**	**1.17**	1.21	1.72	1.94	**1.98**	4.32	5.72	12.03	**14.08**	21.96
risk data unc.	1.09	**1.13**	1.44	**1.80**	2.00	**1.17**	1.21	1.72	1.94	**1.98**	6.50	6.60	12.97	15.28	22.20
risk total unc.	1.14	1.16	1.47	1.85	**1.99**	**1.17**	1.19	1.64	1.94	**1.98**	**3.76**	5.36	**11.15**	14.17	20.65

6.2.6 Robustness to Workload Shifts

A major challenge of using ML-based techniques in query optimization is dealing with the problem of generalization to out-of-distribution test samples. A learned model becomes obsolete if the characteristics of the test query differ significantly from the ones used in training. While model retraining is possible, it takes time, and therefore the optimizer

would typically need to still rely on the obsolete model in the meantime. We evaluate the impact of a workload shift for Roq in comparison to the baselines using two simulations.

In a ***minor workload shift*** simulation, the baseline model is trained using all query templates in the CEB benchmark. The baseline model is tested on making predictions for two selected templates *query025* and *query101*, due to their unique characteristics [4]. The second model is trained using the other 13 query templates in the CEB and tested on the two selected templates.

A ***major workload shift*** scenario is simulated using the CEB and JOB. The baseline model is trained and tested on JOB. This represents an ideal scenario where the test workload is identically distributed as the training workload. The model representing a workload shift is trained on CEB and tested on JOB. This constitutes a major workload shift as the query templates in CEB are entirely different than those in JOB. In addition, our CEB queries have up to 12 joins while JOB has up to 28 joins.

The results shown in Figure 6.4 demonstrate Roq's robustness to out-of-distribution data caused by a *minor workload shift* in the first scenario. Figures 6.4a and 6.4b show that while the baseline models' q-error and correlation degrade significantly with a minor

[4] *query025* is a 7-join query with a complex join graph involving three aliases of the *'DATE_DIM'* table, each used for fine-grained temporal slicing of a different fact table with overlapping time windows. It includes many-to-many joins between the fact tables *STORE_SALES* and *STORE_RETURNS* using multiple join predicates. *query101* is a query with 9 joins, involving a self-join over the *DATE_DIM* table and many-to-many joins between the fact tables *STORE_SALES*, *STORE_RETURNS*, and *WEB_SALES*.

workload shift, Roq's predictive performance degradation is minimal. These results confirm a greater robustness of Roq to out-of-distribution data caused by workload shifts. We also evaluate the impact of workload shift on runtime in the *major workload shift* scenario. As demonstrated in Figures 6.5a and 6.5b, the performance of three out of four baseline models degrades significantly. Lero's performance slightly improves, which can be attributed to the fact that it does not need to make accurate predictions as long as it can properly rank the plans. Roq's base model's performance, also improves compared to the baselines. This can be attributed to the use of GNNs for learning generalizable table and query representations from the CEB workload and use that effectively to make accurate predictions on JOB queries and the fact that our CEB workload includes 1000 queries compared to JOB that includes 113. The additional samples enable a greater performance on an unseen test set when generalizable representations are learned. The Conservative and the Risk strategies, further reduce regressions leading to greater net improvements in runtime. Therefore, Roq and its risk-aware strategies meet the design objective (c) of demonstrating robustness based on well-defined measures.

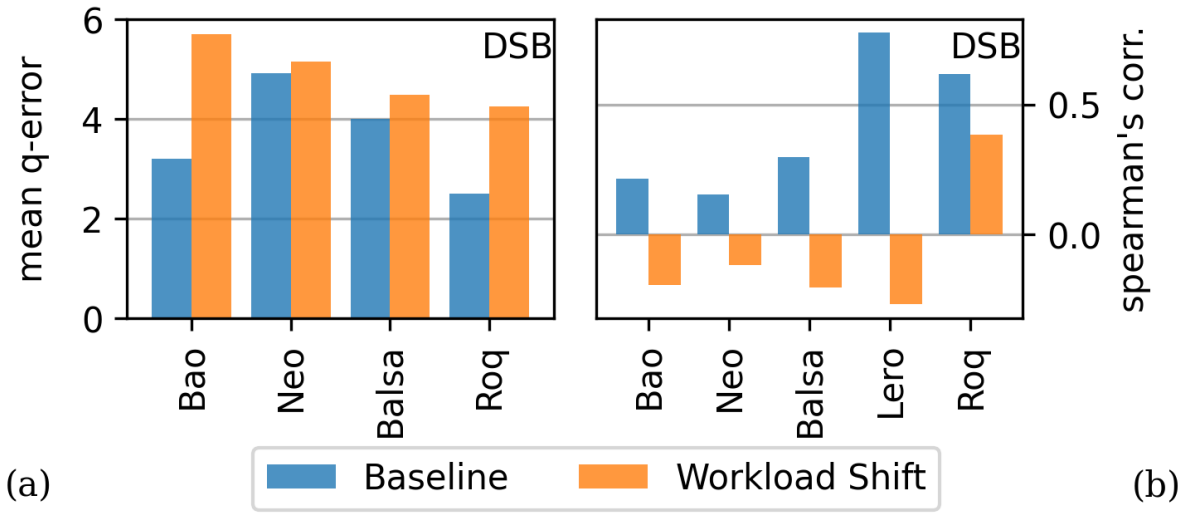


Figure 6.4: Roq's generalization to out-of-distribution samples compared with the baselines. a) average q-error, b) Spearman's correlation with runtime

Figure 6.5: JOB runtime changes after a severe workload shift demonstrated by (a) runtime improvements vs. regressions (b) workload accumulative runtime

6.2.7 Inference Overheads

Capturing estimation risk using the MC Dropout method requires multiple model invocations. A higher number of inferences should provide a more accurate estimate of the expected values and the uncertainties. However, this incurs additional overhead for the query compilation phase, which is on the critical path of query execution and must be

kept at a minimum. We experiment with inference iterations ranging from 5 to 100. We use the suboptimality risk, expected to have a larger overhead compared to the conservative plan selection, given the complex required computations. We capture the inference time, and the runtime and suboptimality of selected plans. We report an average of 10 runs for each experiment. The inference times are captured using a single thread on the CPU, typically available for query compilation. They grow substantially with the number of iterations (Figure 6.6a). Figures 6.6b and 6.6c illustrate the impact of increasing the number of inference iterations on the suboptimality and the runtime of the selected plans. Increasing the number of inferences does not have a major impact on mean suboptimality. The median suboptimality has a sharp decline from 5 to 10 inferences but does not show a significant improvement with larger values. The changes in plan runtimes are not significant either. This means the accuracy provided by 10 inferences should be sufficient for selecting robust plans. This demonstrates that the proposed approach is robust to a limited number of predictions. Figure 6.6d demonstrates the inference overheads of the risk-aware strategies compared with the baselines [5]. This time measurement does not include plan generation as this is identical for all alternatives. Overheads for *Roq-Cons.* and *Roq-Risk* are around 70 and 89 milliseconds respectively. As expected, plan selection by

[5]Balsa and Lero use Neo and Bao's value networks respectively. Therefore, their inference overheads are identical to Neo and Bao. While Lero is a learning to rank model, it does not require multiple inferences to create a total order [179].

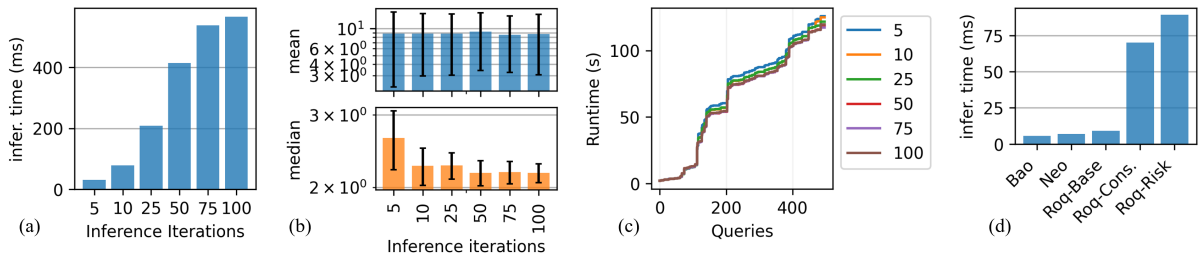


Figure 6.6: The impact of the inference iterations on (a) the inference time, (b) suboptimality, (c) the accumulated runtime of the selected plans and (d) inference overheads for risk-based plan selection methods with 10 inference iterations vs. the baselines

suboptimality risk is slightly more expensive than doing so by the conservative strategy. Note that these measurements are performed based on a prototype code in Python and can be significantly reduced with more optimized implementations. Therefore, Roq and its risk-aware strategies meet the design objective (a) of limiting the compilation overheads at a practical level.

6.3 Overview of Findings

Roq produces predictions with a higher level of accuracy and correlations with runtime. The risk quantification methods and plan selection strategies have significant benefits in selecting robust plans. Our experiments show the role of accounting for plan or estimation risks in the plan selection strategies. The proposed strategies demonstrate significant robustness to shifts in workload characteristics. The quality of the resulting plans is robust

to a limited number of inference iterations used to quantify risks. Therefore, the proposed techniques can be used with minimal inference overheads. Our observations suggest that *Roq-Risk* and *Roq-Cons.* demonstrate similar results. However, from our experience *Roq-Risk.* is preferable as it is non-parametric if the computation overheads can be justified. Also, pruning by plan or estimation risk does not provide significant improvements for these strategies. Lastly, our ablation study suggests that accounting for Plan Risk is more beneficial than accounting for Estimation Risk.

Chapter 7

RobOpt: A Tool for Robust Workload Optimization Based on Uncertainty-Aware Machine Learning

This chapter introduces RobOpt, a practical tool for robust workload optimization that implements the theoretical concepts and architectural designs presented in previous chapters. Developed with a human-centered approach, RobOpt demonstrates the real-world

applicability of uncertainty-aware machine learning techniques for improving database performance and reliability. Section 7.1 presents the overall architecture of RobOpt, illustrating how various components interact to provide a comprehensive workload optimization solution. Section 7.2 provides detailed descriptions of each module, including the Risk-aware Learned Cost Model (RLCM) Trainer for building robust cost models, the Strategy Optimizer for tuning risk-aware plan evaluation parameters, the Workload Optimizer for determining optimal strategies, the Strategy Analyzer for comparative performance analysis, the Workload Analyzer for identifying problematic queries, and the Workload Planner for strategy selection automation. Section 7.2.6 illustrates three practical use case scenarios that demonstrate RobOpt's capabilities in strategy comparison, workload analysis, and comprehensive workload planning. Section 7.4 concludes by discussing how RobOpt addresses the specific requirements of different user personas—database developers, administrators, and end-users—as identified in Chapter 1. Through its intuitive interface and powerful optimization capabilities, RobOpt bridges the gap between theoretical research and practical database management, making robust query optimization accessible to both novice users and experienced database professionals.

7.1 Architecture

The RobOpt architecture, shown in Figure 7.1, comprises modules, each designed to offer specific functionalities to the user. The user can upload a workload via an intuitive interface, choose among various plan selection strategies, view detailed analysis results, and perform robust workload planning. The modular design of RobOpt allows for seamless integration and independent upgrades of each component, ensuring scalability and maintainability in diverse database environments.

The Workload Manager integrates with an RLCM based on the architecture proposed in Chapter 5. This includes the Query Encoder, Plan Encoder, and Estimator modules. The Plan Encoder leverages a Graph Neural Network (GNN) to transform the join graph representation of the query into meaningful query embeddings. Additionally, a Tree Convolutional Neural Network (TCNN) processes the plan tree to derive plan embeddings. The estimator then concatenates these embeddings and uses a series of Multilayer Perceptrons (MLP) to predict the expected execution time and quantify both data and model uncertainties. This joint embedding space enables a fine-grained, uncertainty-aware cost estimation that is crucial for robust optimization.

The input workload is received by the Planner, which utilizes the *RLCM Trainer* to adapt the cost model to the current workload characteristics. This involves generating

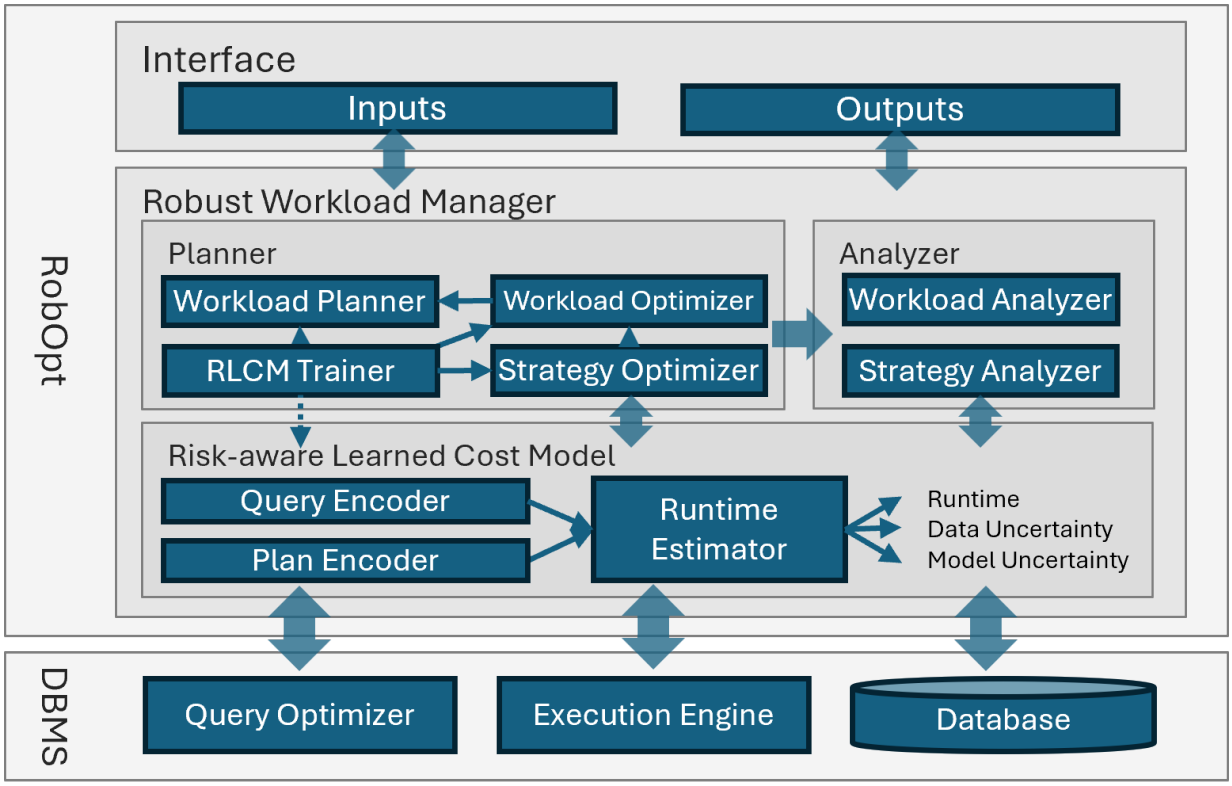


Figure 7.1: RobOpt Workload Manager Architecture; its sub-modules, and interactions with the query optimizer, execution engine, and the user interface

training data and retraining the RLCM. The *Strategy Optimizer* then fine-tunes the parameters of the risk-aware strategies. Following this, the *Workload Optimizer* evaluates the performance of the optimized strategies on a validation set and determines the best strategy for each query. Finally, the *Workload Planner* applies the optimal strategy for future query executions by matching incoming queries with the pre-computed *Query-Strategy* Profile. An offline module is also available for in-depth analysis of both strategies and workload characteristics. In the following, we describe the functionalities enabled by each module.

7.2 Description of Modules

RopOpt's Workload Manager includes a Planner, an Analyzer, and an RLCM. The RLCM is introduced and discussed in detail in chapter 5 and will not be repeated here. The following sections describe the sub-modules of Planner (including RLCM Trainer, Strategy Optimizer, Workload Optimizer, and Workload Planner) and Analyzer (including Strategy Analyzer and Workload Analyzer).

7.2.1 RLCM Trainer

This module is responsible for building a robust cost model tailored to the workload at hand. Data is collected from historical workload logs, which consist of queries executed against the RDBMS. These queries are transformed into templates by replacing literal values with parameter markers, ensuring that the training process generalizes across different query instances. Thousands of query instances are generated by binding these markers with randomly sampled values from the corresponding database columns, thus augmenting the training data.

For each generated query, multiple execution plans are produced by varying the hint sets that steer the underlying *Query Optimizer*—for example, by toggling join or access

operators. The runtime for every query-plan pair is then recorded. This rich dataset of query-plan pairs, alongside the observed execution times, serves as both the training and validation sets for the RLCM. During training, the Query Encoder and Plan Encoder generate embeddings that capture the structural properties of queries and plans, respectively. The estimator then fuses these embeddings to predict not only the expected execution time but also to estimate the associated data and model uncertainties. This uncertainty-aware prediction is essential for making risk-averse decisions in later modules.

7.2.2 Strategy Optimizer

The Strategy Optimizer leverages the validation set to fine-tune the parameters of risk-aware plan evaluation strategies. It systematically explores a multi-dimensional parameter space via random search techniques, evaluating combinations based on two main criteria: minimizing execution runtime and reducing suboptimality. Suboptimality here is defined as the deviation from the best-known execution plan, quantified through statistical measures such as the mean and upper percentiles.

For each strategy, the optimizer records the best-performing parameter combination and logs the corresponding runtime and suboptimality metrics in a *Strategy Performance Profile*. This profile not only serves as a reference for future inferences but also provides insights into the sensitivity of each strategy to different workload characteristics. This sys-

tematic exploration ensures that the risk-aware strategies are both effective and adaptable to varying query complexities and data distributions.

7.2.3 Workload Optimizer

The Workload Optimizer takes the refined validation set and determines the optimal strategy for executing each query. It evaluates each query against all available strategies and selects the one that minimizes either the runtime or suboptimality, in line with the user's performance criteria. Once the best strategy is identified, the query is encoded and paired with its corresponding optimal strategy in the *Query-Strategy Profile*.

An essential feature of this module is its visualization capability. Query embeddings, which encapsulate the semantic and structural characteristics of the queries, are projected into a low-dimensional space using techniques such as PCA, shown in Figure 7.2. This visual mapping allows users to see clusters of queries that are best optimized by a given strategy, offering insights into the inherent similarities between different workload patterns. The graphical interface aids in understanding how specific strategies perform across various types of queries, thereby enhancing the overall decision-making process.

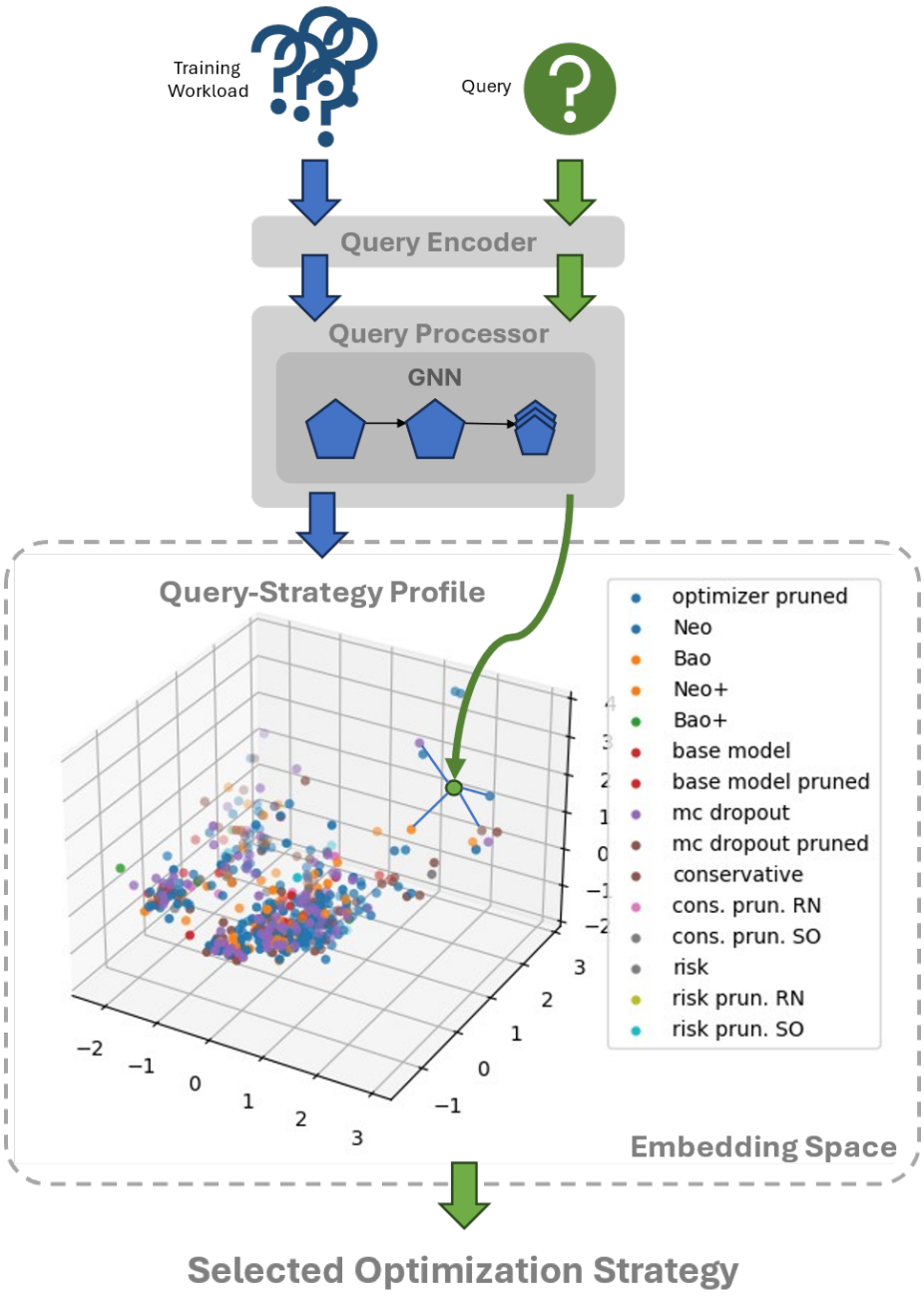


Figure 7.2: Mapping an unseen query to the embedding space of the training queries to determine the optimal strategy based on the nearest neighbors

7.2.4 Strategy Analyzer

The Strategy Analyzer delves into the *Strategy Performance Profile* to create comprehensive *Strategy Risk Profiles* for each plan evaluation strategy. By aggregating performance data, this module computes key metrics such as total runtime and robustness. Robustness is quantified using high-percentile measures (e.g., the 95^{th} or 99^{th} percentile of suboptimality), which highlight the worst-case performance scenarios.

These metrics are then visualized in a 2-D scatter plot, where one axis represents runtime and the other captures robustness. This plot helps to distinguish strategies that offer excellent average performance from those that may occasionally yield suboptimal plans due to outliers. Users can interactively explore the trade-offs between speed and reliability, enabling them to select strategies that best align with their operational risk tolerance. The visual insights provided by the Strategy Analyzer are critical for both immediate decision-making and long-term strategic planning.

7.2.5 Workload Analyzer

The Workload Analyzer module is designed to provide an in-depth assessment of any workload by leveraging the *Strategy Performance Profile*. It systematically identifies problematic queries that exceed predefined thresholds in terms of suboptimality, data uncertainty,

or model uncertainty (by default, the 95^{th} percentile is used). The resulting *Workload Risk Profile* highlights queries that are at risk of underperformance.

This module employs advanced pattern recognition techniques to uncover common characteristics among high-risk queries. Such insights can include correlations between specific query structures and elevated uncertainty metrics, or trends indicating when certain database operations lead to unpredictable runtimes. By offering both high-level overviews and granular drill-downs, the Workload Analyzer assists database administrators in pinpointing areas for performance tuning or further investigation, thus enhancing overall workload planning and resource allocation.

7.2.6 Workload Planner

The Workload Planner module finalizes the optimization process by mapping the input workload to the best available strategies using the pre-computed *Query-Strategy Profile*, as shown in Figure 7.2. Each incoming query is first encoded and then compared to the stored embeddings using proximity measures such as Cosine similarity or Euclidean distance. The strategy associated with the nearest neighbor in the profile is then selected for that query.

This module not only automates strategy selection but also incorporates a feedback loop whereby an advanced user can override the default choice based on additional risk analysis

at the query level. Such manual adjustments allow for fine-grained control, ensuring that unique or outlier queries are handled appropriately. Finally, the optimized queries, along with their respective execution plans, are forwarded to the *Execution Engine* for actual deployment. This seamless integration guarantees that the entire process—from query submission to execution—is both efficient and robust.

7.3 Use Case Scenarios

Three use case scenarios of RobOpt are discussed in the following sections using realistic workload planning scenarios with IBM Db2 as the underlying database engine. Each scenario addresses the needs of specific user personas identified in Section 1.1: database developers benefit from the Strategy Comparison scenario by evaluating different optimization approaches without manual reconfiguration; database administrators leverage the Workload Analysis scenario to automate the identification of problematic queries and minimize maintenance overheads; and end-users' experience is enhanced through the Workload Planning scenario, which ensures consistent performance by minimizing regressions while improving average query response times. These scenarios demonstrate how RobOpt provides value across different stakeholder perspectives, from technical implementation to operational management and end-user experience. The evaluation is structured into the

following scenarios.

7.3.1 Comparison of Strategies

This scenario showcases how different plan evaluation strategies can be compared based on their performance and risk profiles. A sample workload is loaded, and multiple risk-aware strategies are selected. RobOpt processes the workload, displaying a visualization that summarizes the runtime and robustness metrics for each strategy, as illustrated in Figure 7.3. The interactive interface allows users to zoom in on clusters of strategies and analyze trade-offs, making it clear how workload characteristics influence strategy performance. This hands-on experience underscores the importance of balancing average performance with risk mitigation, as certain strategies may perform well under normal conditions but falter in the presence of outliers.

7.3.2 Workload Analysis

In this scenario, a database administrator is guided through the process of analyzing a workload to detect long-running or high-risk queries. The process begins with a novice user loading a sample workload into the system. RobOpt then provides a high-level overview of the workload's performance, highlighting key metrics such as overall runtime distribution

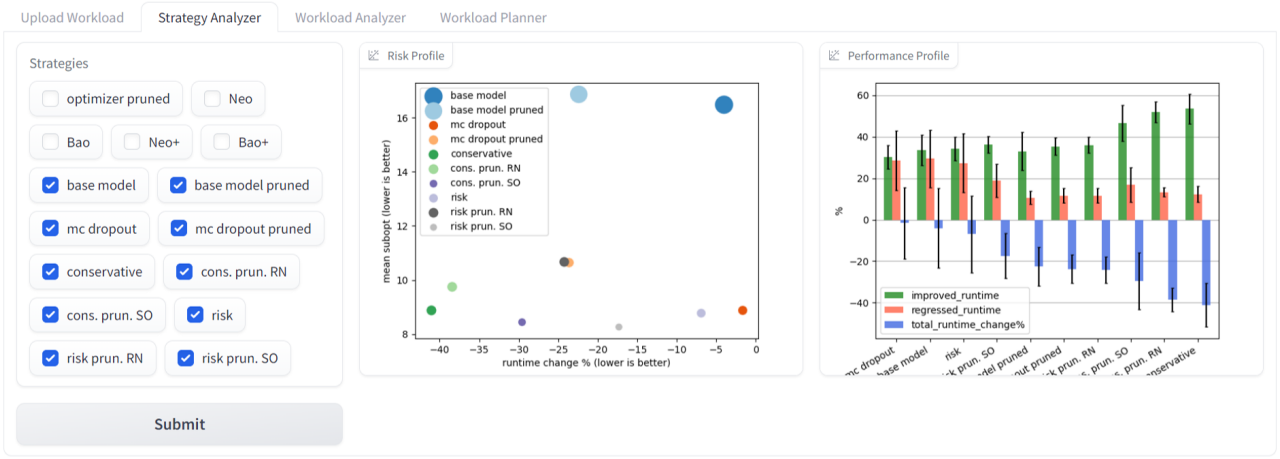


Figure 7.3: The user interface of RobOpt Strategy Analyzer, with interactive plots and metrics

and uncertainty levels across different query types. For advanced users, the tool offers deeper analytical capabilities—such as drilling down into individual queries, comparing uncertainty metrics, and visualizing the impact of specific operations on suboptimality. The visualizations provided by the Workload Analyzer (see Figure 7.4) help users identify problematic patterns and derive insights into which queries might require manual tuning or alternative planning strategies.

7.3.3 Workload Planning

This scenario illustrates the complete process of workload planning using a test workload. The process is presented sequentially: starting with the Strategy Optimizer tuning the risk-aware strategies, followed by the Workload Optimizer matching queries with optimal

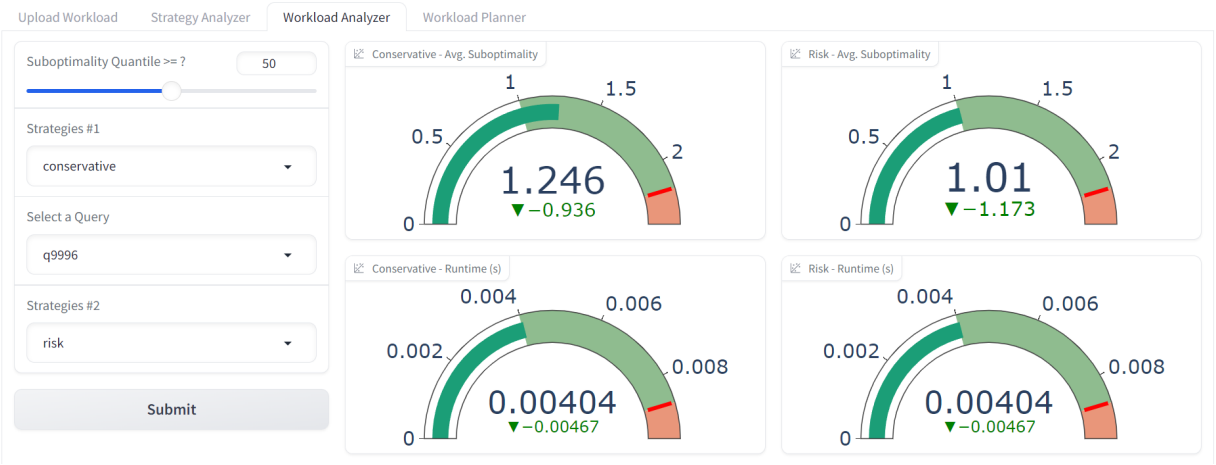


Figure 7.4: The user interface of RobOpt Workload Analyzer

strategies, and culminating in the Workload Planner applying these strategies in real-time. As depicted in Figure 7.5, the process involves encoding each query, determining its nearest neighbor in the Query-Strategy Profile, and assigning the corresponding optimal strategy. An advanced user can interact with the interface to override the default selections based on additional risk insights. This interactive functionality not only validates the efficacy of the automated process but also demonstrates RobOpt's ability to integrate with prior systems such as Neo [101] and Bao [100], thereby extending traditional workload optimization methods with rigorous risk quantification to minimize regressions.

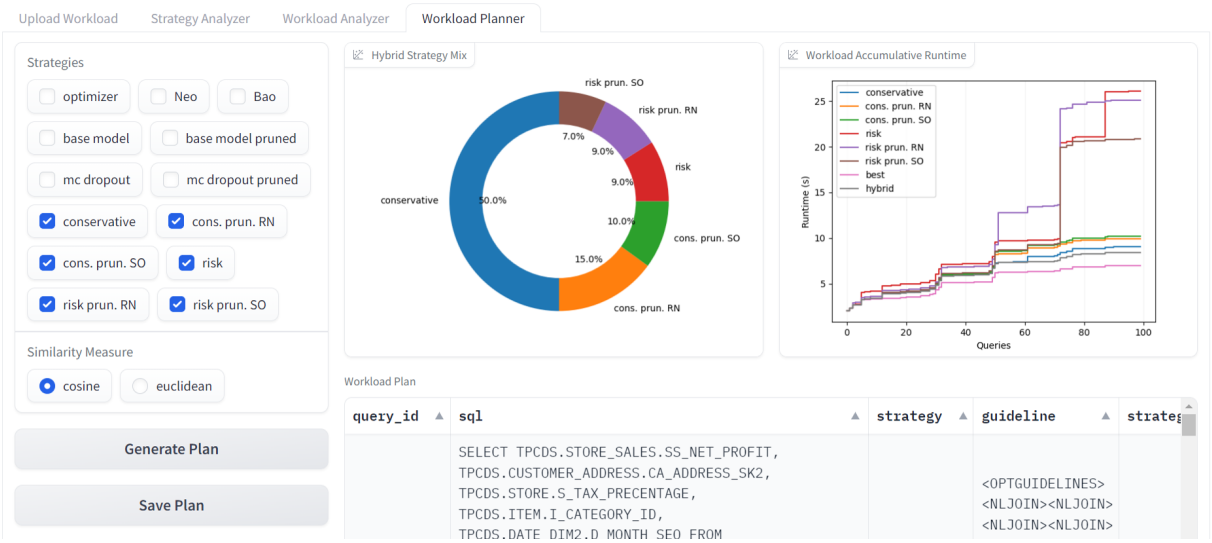


Figure 7.5: The user interface of RobOpt Workload Planner

7.4 Discussion on Meeting the Requirements of the User Personas

Section 1.1 reviewed the requirements for the modernization of a query optimizer from the perspective of three personas: The database developer, the database administrator, and the end user. The following sections review those requirements and discuss how RobOpt meets them.

7.4.1 Database Developer

Section 1.1 explained that the modernization of the query optimizer requires the following from the perspective of the database developers:

1. Replacing the heuristics-based components that require human-led reconfiguration as new system architecture and hardware emerge, with counterparts that adapt to the changes automatically

2. Having minimal configuration and tuning overheads such that the benefit of their adoption for the development teams outweighs their costs

3. As much as possible, targeting modules with minimal cascading effects on the other modules of the query optimizer to minimize the risks of instability

RobOpt meets the first requirement by replacing the query optimizer's cost model with an ML-based counterpart that does not require crafting various cost model parameters as new computing environments emerge. Instead, it learns the system's behavior from query execution feedback through the RLCM and automatically adapts to changing hardware specifications and system configurations without manual intervention.

RobOpt meets the second requirement by enabling the adoption of different optimization strategies and obtaining optimal configurations automatically without the need for

expert input. The Strategy Optimizer module systematically explores the parameter space and fine-tunes strategies based on actual workload characteristics, eliminating the need for manual tuning. Additionally, the Workload Planner automates strategy selection for incoming queries by matching them to pre-computed Query-Strategy Profiles, reducing ongoing maintenance efforts.

RobOpt meets the third requirement by adopting a modular design where no changes are needed in the query execution engine, and changes to the query optimizer are limited to augmenting the database statistics with the additional statistics needed to encode the queries and plans. The integration is designed to be minimally intrusive, allowing RobOpt to function as a plug-in component that enhances the existing optimizer without disrupting its core functionality or risking system stability.

7.4.2 Database Administrator

Section 1.1 explained that the modernization of the query optimizer requires the following from the perspective of database administrators:

1. Automating database maintenance tasks as much as possible
2. Making the databases adapt to the changing data, workload, and computing environment automatically

3. Adopting methods and configurations that achieve an acceptable average performance while minimizing the risk of regressions

RobOpt addresses the first requirement by automating key database maintenance tasks through its Workload Manager. The RLCM Trainer automatically handles data collection from historical workload logs, generates training data by creating thousands of query instances with different parameter bindings, and retrains the cost model as needed. The Workload Analyzer further automates the identification of problematic queries that exceed performance thresholds, eliminating the need for manual monitoring and analysis.

RobOpt meets the second requirement through its adaptive learning capabilities. The system continuously updates its cost model based on actual execution feedback, automatically adapting to changes in data distribution, query patterns, and computing resources. The Strategy Optimizer regularly recalibrates plan evaluation strategies to maintain optimal performance under varying conditions, ensuring that the system remains effective as the environment evolves.

RobOpt addresses the third requirement by implementing risk-aware optimization strategies that explicitly balance average performance with robustness. The system provides database administrators with visibility into performance trade-offs through interactive visualizations in the Strategy Analyzer, allowing them to select strategies that minimize per-

formance regressions while maintaining competitive average execution times. The tool's ability to quantify and manage uncertainty ensures more predictable performance across diverse workloads.

7.4.3 End-user

Section 1.1 explained that the modernization of the query optimizer requires the following from the perspective of end-users and business perspectives:

1. Improving the average performance, while minimizing the risk of regressions to the extent possible

2. Not adversely impacting the system's overall stability

RobOpt addresses the first requirement by implementing risk-aware plan selection strategies that significantly reduce performance regressions while maintaining or improving average query execution times. Through its uncertainty quantification framework, RobOpt selects plans that minimize the risk of catastrophic performance degradation, which is crucial for maintaining a consistent user experience. The experimental results demonstrate that RobOpt improves the 99th percentile of suboptimality by up to 44.4% compared to baseline methods, effectively preventing the severe performance cliffs that can frustrate end-users (refer to the details in section 6).

RobOpt meets the second requirement by maintaining system stability through its conservative approach to plan selection and its modular design that isolates risk-aware optimization from core database functionality. The system's risk quantification mechanism ensures that uncertainty-aware decisions are made transparently without introducing unpredictable behavior. Furthermore, the minimal compilation-time overhead (70-89 milliseconds) ensures that query optimization remains efficient and does not noticeably increase latency for end-users. By providing more predictable query performance, RobOpt enhances the overall user experience, which is particularly important for applications with strict service-level agreements or real-time requirements.

Chapter 8

Conclusion

This chapter concludes the thesis by synthesizing the theoretical, architectural, and experimental contributions to robust query optimization, while reflecting on their implications for the field of database management systems. The research presented, guided by the Design Science Research methodology detailed in Chapter 2, has addressed the critical challenge of robustness in query optimization by proposing a principled approach that leverages approximate probabilistic machine learning techniques to quantify and mitigate risks associated with query execution plans. Section 8.1 summarizes the key research contributions, highlighting the novel theoretical framework for uncertainty decomposition, the risk-aware plan evaluation strategies, and the practical implementation through the RobOpt tool. Section 8.3 provides a critical analysis of the experimental results, interpreting their signif-

icance in the context of existing approaches. Section 8.4 explores the broader implications of this work for database management systems, particularly how risk-aware optimization can transform commercial systems. Section 8.5 acknowledges the limitations of the current approach, while Section 8.6 outlines promising directions for future research that build upon this foundation. The chapter concludes in Section 8.7 with final remarks on the significance of embracing uncertainty rather than ignoring it in query optimization. Together, these sections provide a comprehensive reflection on how the proposed methods advance the state-of-the-art while opening new avenues for research in robust database systems.

8.1 Summary of Research Contributions

The primary objective of this research has been to formalize the notion of robustness in query optimization and develop techniques that enable robust plan evaluation and selection. Traditional query optimizers predominantly focus on expected optimality—selecting plans that minimize estimated costs—without accounting for the inherent uncertainties in these estimates. This approach often leads to suboptimal performance in real-world execution scenarios where estimation errors and environmental variations are inevitable.

Our key theoretical contributions include:

- A rigorous mathematical framework that decomposes cost model uncertainty into

distinct components: plan risk (uncertainty inherent to the plan structure and sensitivity to parameter variations) and estimation risk (uncertainty in cost estimates due to limited knowledge of the ideal cost model).

- A formal definition of suboptimality risk as the likelihood of a plan being suboptimal compared to alternatives, quantified through probabilistic methods that capture both aleatoric and epistemic uncertainties.

- Novel risk-aware plan evaluation strategies that utilize uncertainty quantification to select robust plans, including plan selection by suboptimality risk, conservative plan selection, and search space pruning based on risk assessments.

Technical innovations realized in this research include:

- The Roq architecture, which leverages Graph Neural Networks to learn query and plan representations that capture complex structural characteristics and their impact on execution costs.

- A dual-branch estimation module that simultaneously predicts execution times and their associated uncertainties, enabling comprehensive risk quantification.

- RobOpt, a practical workload optimization tool that demonstrates the real-world applicability of our theoretical framework by providing interactive analysis and robust

optimization of database workloads.

8.2 Addressing Robustness Characteristics

The characteristics of robustness outlined in the introduction represent conceptual descriptions of robustness for database systems in general, as offered by prior work [52]. This work acknowledges that robustness is application-dependent, and the specific characteristics of robustness may vary based on the context. That is why this thesis proposed a new formalization for robustness based on plan risk, estimation risk, and suboptimality risk. That said, even these conceptual characteristics can be mapped to the proposed approach.

- **Bounded Maximum Suboptimality:** Roq explicitly quantifies suboptimality risk, which measures the probability of a plan being worse than its alternatives at runtime. By incorporating this risk into plan selection, this approach ensures that selected plans avoid catastrophic regressions, effectively bounding maximum suboptimality. This is validated by experiments demonstrating lower tail-end suboptimality, as shown in the experimental results where risk-aware strategies significantly improved the 99th percentile of suboptimality by 44.4% compared to ML-based baselines.

- **Graceful Performance Degradation:** Through risk-aware strategies and probabilistic modeling of plan and estimation uncertainties, Roq favors plans whose performance remains stable under cost estimation errors or workload shifts. This leads to smooth degradation rather than performance cliffs, as demonstrated in experiments with workload shifts where Roq maintained relatively stable performance while baseline methods exhibited significant degradation.

- **Scalability:** The proposed learned cost model architecture, based on Graph Neural Networks with uncertainty quantification, generalizes across diverse benchmarks (JOB, CEB, DSB, TPC-DS+). It demonstrates robustness when query complexity and workload distribution change, highlighting the approach's scalability. The experimental evaluation across multiple benchmarks with varying complexity and characteristics validates this scalability characteristic.

- **Theoretical Guarantees:** While this work establishes a theoretical framework for risk decomposition (plan risk, estimation risk, suboptimality risk), the current implementation does not provide formal mathematical guarantees in the strict sense. Instead, it provides a principled probabilistic formulation with empirical validation. This represents a limitation acknowledged in this thesis and outlined as future work, particularly in developing more rigorous theoretical foundations for the proposed risk

quantification methods.

The mapping of these characteristics to the proposed approach demonstrates that while the specific formalization differs from the original conceptual framework, the underlying principles of robustness are preserved and enhanced through the quantitative risk-based methodology developed in this thesis.

8.3 Critical Analysis of Results

Our extensive experimental evaluation, conducted following the rigorous evaluation strategy outlined in Chapter 2, has yielded several significant findings that validate the efficacy of our approach:

The Roq base model consistently outperformed state-of-the-art baselines in terms of prediction accuracy, with lower q-errors and higher correlation with actual execution times. This superior predictive performance stems from our innovative model architecture, which employs GNNs to process query join graphs and capture intricate relationships between tables and predicates.

Risk-aware plan selection strategies demonstrated substantial improvements in robustness metrics. Specifically, plan selection by suboptimality risk (Roq-Risk) improved the

99th percentile of suboptimality by 44.4% compared to ML-based baselines, while conservative plan selection (Roq-Cons) achieved comparable improvements. These strategies effectively reduced the occurrence of severe performance degradation cases, confirming their ability to enhance robustness.

The ablation study investigating the impact of different uncertainty types revealed that both plan risk (data uncertainty) and estimation risk (model uncertainty) contribute significantly to improved performance. Utilizing both uncertainty types in combination yielded optimal results, suggesting a synergistic effect in risk quantification. Notably, plan risk showed a more pronounced impact on runtime performance, effectively preventing the selection of plans with excessive execution times.

Our analysis of workload shifts demonstrated Roq's exceptional resilience to distribution changes. Under both minor and major workload shifts, Roq maintained relatively stable performance, while baseline methods exhibited significant degradation. This robustness can be attributed to Roq's ability to learn generalizable representations through its GNN-based architecture, enabling effective transfer learning across different query workloads.

Regarding computational overhead, our investigation into inference iterations revealed that as few as 10 iterations were sufficient to achieve robust plan selection, with minimal gains observed beyond this threshold. This finding is particularly important for practical

implementations, as it suggests that the benefits of risk-aware optimization can be realized without prohibitive compilation-time overheads.

8.4 Implications for Database Management Systems

The research presented in this thesis has several profound implications for the field of database management systems:

- **Paradigm Shift in Query Optimization:** Our work challenges the traditional focus on expected optimality by introducing risk-awareness as a fundamental consideration in plan selection. This represents a paradigm shift that aligns query optimization more closely with real-world performance objectives.

- **Integration of Machine Learning and Database Systems:** The successful application of advanced machine learning techniques, particularly approximate probabilistic methods, to query optimization demonstrates the potential for deeper integration between these fields. This integration can address longstanding challenges in database systems through novel computational approaches.

- **Practical Tools for Database Administrators:** RobOpt exemplifies how theoretical advances can be translated into practical tools that assist database admin-

istrators in workload analysis and optimization. Such tools can significantly reduce the expertise required for database tuning and maintenance.

- **Enhanced Predictability of Database Performance:** By quantifying and mitigating risks, our approach contributes to more consistent and predictable database performance, which is critical for applications with strict service-level agreements or real-time requirements.

Commercial database systems could benefit substantially from incorporating risk-aware optimization techniques. The ability to identify and mitigate potential performance cliffs would enhance system stability, reduce the need for manual intervention, and improve overall user satisfaction. Furthermore, the risk quantification framework could be extended to other aspects of database management, such as resource allocation, concurrency control, and data partitioning.

8.5 Limitations of the Current Work

Despite the promising results demonstrated by our robust query optimization approach, several limitations warrant acknowledgment and detailed discussion:

- **Computational Overhead:** While we demonstrated that 10 inference iterations

are sufficient for effective risk quantification, this still introduces additional computational costs compared to traditional optimization methods. The experiments in Section 6.2 revealed inference overheads of approximately 70 milliseconds for conservative plan selection and 89 milliseconds for selection by suboptimality risk. In highly concurrent environments with thousands of queries per second, even these modest overheads could impact system throughput. The overhead becomes particularly concerning for complex queries with numerous join operations, as the size of the vectorized plan representations grows, increasing the computational demands of both the Graph Neural Network and Tree Convolutional Neural Network components. Additionally, as shown in Figure 6.6, while the benefits of risk-aware optimization plateau after approximately 10 inference iterations, production environments with stricter latency requirements might need to further reduce this number, potentially sacrificing some robustness benefits.

- **Training Data Requirements:** The effectiveness of our approach depends heavily on the availability of representative training data. As detailed in Section 6.1, collecting execution statistics involves generating multiple plans for each query using various hint sets, executing each plan, and measuring its runtime. This process is resource-intensive, particularly for new database deployments or rapidly evolving schemas. The quality of the training data also significantly impacts model perfor-

mance. Our experimental setup used 13 carefully selected hint sets to generate diverse yet reasonable execution plans. However, as noted in Section 6.1, this doesn't fully represent how the optimizer traverses the search space in practice. Furthermore, timeout thresholds were needed to terminate excessively long executions, potentially limiting the model's exposure to extremely suboptimal plans and affecting its ability to accurately quantify their risks.

- **Independence Assumption:** Our formulation of suboptimality risk in Section 4.2 assumes independence between the cost distributions of alternative plans. As acknowledged in Section 4.2, this assumption results in overestimation of risk by increasing the variance of the cost difference between plans. While we demonstrated that this simplification still yields significant improvements in plan selection robustness, a more accurate modeling of joint distributions could further enhance performance. The independence assumption was made primarily for computational tractability, as calculating pairwise joint distributions for all plans in the search space would be prohibitively expensive. The thesis demonstrates in Equation 4.22 that vectorization techniques help mitigate computation overhead, but capturing plan correlations remains challenging, especially when the number of candidate plans is large or when plans share common substructures.

- **Black-Box Nature of Deep Learning Models:** The neural network components in our architecture, particularly the GNN-based Query Processor and TCNN-based Plan Processor described in Chapter 5, lack the interpretability of traditional cost models. While effective at capturing complex relationships, these models operate as black boxes, making it difficult to understand why certain plans are deemed risky or how specific query structures influence uncertainty estimates. This opacity may impede adoption in environments where explainability is paramount, such as mission-critical systems or regulated industries. Unlike traditional optimizers where the cost model can be traced and debugged, errors or unexpected behaviors in our approach are more challenging to diagnose and correct. This limitation is particularly relevant in the context of database systems, where predictability and understanding of system behavior are often valued as highly as performance.

- **Adaptation to System Changes:** While our approach demonstrated robustness to workload shifts in Section 6.2, adaptation to significant hardware or configuration changes might require model retraining. The experimental results show impressive resilience when faced with different query templates (as in the CEB to JOB transfer), but changes in underlying system characteristics, such as CPU architecture, memory hierarchy, or storage technology, could invalidate the learned cost relationships. The RLCM Trainer component of RobOpt described in Section 7.2 provides capabilities

for retraining models as system conditions evolve, but this process still requires collection of new training data and introduces maintenance challenges. Furthermore, the thesis doesn't extensively evaluate how frequently retraining might be necessary in dynamic environments, such as cloud deployments with variable resource allocations or systems that undergo regular hardware upgrades.

- **Scaling to Very Large Schemas and Complex Queries:** Although our experiments included queries with up to 28 joins in the JOB benchmark, there may be challenges when scaling to even larger schemas or more complex query patterns. The size of the query and plan encodings grows with the number of tables and joins, potentially leading to memory constraints and increased training and inference times. Additionally, as noted in Section 5.1, the edge feature encoding captures a rich set of join characteristics, which might become unwieldy for extremely complex schemas with numerous join relationships. The representation learning approach might also face challenges with highly specialized query patterns not well-represented in the training data, such as recursive queries, complex aggregations, or queries involving user-defined functions. These limitations could restrict the applicability of our approach in certain specialized database applications with unique workload characteristics.

8.6 Future Research Directions

The work presented in this thesis opens up numerous avenues for future research:

- **Hierarchical Uncertainty Quantification:** Extending the risk quantification framework to operate at multiple granularities—from individual operators to subplans to complete query plans—could provide more nuanced insights and enable more targeted optimizations. Currently, as described in Chapter 5, our approach quantifies uncertainties at the plan level, but finer-grained analysis could reveal which specific components of a plan contribute most significantly to its overall risk profile. This hierarchical approach would build upon our cost model uncertainty decomposition (Equation 4.1) by applying similar principles to subplan components. By identifying high-risk operators or join sequences, the optimizer could focus on generating alternatives specifically for these problematic sections while retaining the less risky portions, potentially leading to more efficient search space exploration. Additionally, quantifying uncertainty at the operator level could enable more precise tuning of the system's statistics collection mechanisms, focusing resources on high-impact areas rather than applying uniform efforts across all database objects.

- **Online Learning and Adaptation:** Developing techniques for continual learning

that adapt to changing workloads and system conditions without requiring extensive retraining would enhance the practical utility of our approach. While Section 6.2 demonstrated Roq's robustness to workload shifts, a fully adaptive system would continuously update its models based on execution feedback, making incremental adjustments rather than requiring periodic complete retraining. This direction would involve designing efficient algorithms for incrementally updating the learned representations within the GNN and TCNN components described in Chapter 5, potentially using techniques such as transfer learning or active learning to minimize the amount of new training data required. Critical challenges include balancing adaptation speed against stability, preventing catastrophic forgetting of previously learned patterns, and ensuring that online updates do not introduce additional compilation overheads. Success in this area would significantly reduce the maintenance burden associated with ML-based optimization systems, addressing one of the key limitations discussed in Section 8.5.

- **Multi-objective Optimization:** Incorporating additional objectives beyond execution time, such as memory consumption, energy efficiency, or resource fairness, would enable more holistic optimization strategies that balance multiple competing concerns. Our current formulation of suboptimality risk (Section 4.2) focuses exclusively on execution time, but modern database systems, particularly those in shared

environments, must consider broader resource implications. This extension would require reformulating the risk quantification framework to account for multiple dimensions of uncertainty, potentially using techniques from multi-objective Bayesian optimization. The strategy optimizer component of RobOpt (Section 7.2) would need enhancement to explore the resulting Pareto frontier of plans that represent different trade-offs between objectives. This approach would be particularly valuable in cloud environments where costs are directly tied to resource consumption, or in battery-powered edge computing scenarios where energy efficiency is paramount alongside performance considerations.

- **Cross-Engine Generalization:** Investigating transfer learning approaches that enable models trained on one database system to generalize to others would reduce the implementation burden and accelerate adoption. The model architecture described in Chapter 5 is tailored to the specific query and plan representations of our experimental system, but the underlying principles of uncertainty quantification could apply across different database engines. Research in this direction would focus on developing more abstract representations of queries and plans that capture the essential characteristics influencing performance while abstracting away engine-specific details. This might involve pretraining on data from multiple systems to learn universal performance patterns, then fine-tuning with limited data from the target system.

Successful cross-engine generalization would dramatically reduce the barrier to adoption for robust query optimization, allowing smaller database projects or specialized systems to benefit from models trained on more extensively instrumented platforms.

- **Integration with Federated Learning:** Exploring privacy-preserving techniques that enable collaborative model training across organizations without sharing sensitive query workloads could address data scarcity challenges. As noted in Section 8.5, the effectiveness of our approach depends on the availability of substantial training data, which may be limited in new deployments or specialized domains. Federated learning would allow multiple organizations to collectively train more robust models by sharing model updates rather than raw query data. This approach would require adapting the RLCM architecture (Chapter 5) to support decentralized training while maintaining performance guarantees. Additional research challenges include handling the heterogeneity of database environments across participants, preventing poisoning attacks where malicious participants attempt to sabotage the global model, and developing fair incentive mechanisms for participation. Successfully addressing these challenges would enable organizations to benefit from collective optimization knowledge while maintaining the confidentiality of their specific workloads and data patterns.

- **Uncertainty-Aware Query Compilation:** Extending risk-awareness beyond plan selection to influence the generation of adaptive execution plans that can dynamically adjust to runtime conditions based on predicted uncertainties would represent a fundamental advancement. Currently, our approach selects from a set of fixed plans generated by the traditional optimizer (Section 6.1), but a more integrated approach could generate plans specifically designed for robustness. This research direction would involve modifying the plan generation process to incorporate uncertainty predictions directly, potentially introducing new physical operators designed for adaptability or creating contingency paths within execution plans. For example, highly uncertain join operations could be compiled with runtime monitoring and fallback strategies, while more predictable operations could use more efficient but less flexible implementations. This deeper integration with the compilation process would require significant architectural changes to traditional query optimizers but could yield substantial robustness improvements by considering uncertainty from the earliest stages of plan generation rather than only during plan selection.

- **Incorporating System Resource Dynamics:** Enhancing the model to account for concurrent workloads, resource contention, and system load variations would improve robustness in multi-tenant environments. Our current approach, as discussed in Chapter 5, models uncertainties related to plan structure and cost estimation but

does not explicitly consider the dynamic nature of system resources during execution. Future research could extend our risk quantification framework to incorporate runtime resource variability, modeling how execution times are affected by fluctuations in available memory, CPU scheduling, or I/O bandwidth. This would require collecting and encoding additional features related to system state, potentially using techniques from time series forecasting to predict resource availability during query execution. The Query Processor component (Section 5.3.1) would need enhancement to incorporate these contextual features alongside the structural characteristics of the query itself. Successfully modeling these dynamics would be particularly valuable in cloud environments with shared resources or systems with highly variable workloads, where execution conditions can change significantly between optimization and execution time.

- **Explainable Risk Quantification:** Developing techniques to explain the sources of uncertainty in cost predictions would enhance user trust and facilitate debugging of problematic queries. As noted in Section 8.5, the black-box nature of deep learning models limits their interpretability, making it difficult to understand why certain plans are classified as risky. This research direction would focus on methods to decompose and visualize the contributions of different query elements to overall uncertainty, potentially using techniques such as integrated gradients or attention visualization

adapted to graph structures. The goal would be to provide database administrators with actionable insights about which specific aspects of a query (e.g., certain join predicates, data skew in particular tables, or correlation assumptions) contribute most significantly to uncertainty, enabling targeted improvements to statistics or query reformulation. This explainability would bridge the gap between traditional cost models with transparent calculations and the more powerful but opaque learned models, potentially accelerating adoption in environments where interpretability is a key requirement.

- **Hardware-Aware Optimization** Investigating techniques to make risk-aware optimization cognizant of the specific hardware characteristics of the execution environment could improve performance and reliability. While our current approach implicitly learns hardware behavior through training data collected on the target system, a more explicit incorporation of hardware parameters could enable better generalization and adaptation. Future research in this direction would involve developing hardware-aware features that capture the critical performance characteristics of different CPU architectures, memory hierarchies, storage technologies, and networking configurations. These features could be incorporated into both the query and plan encodings (Sections 5.1 and 5.2), enabling the model to learn how hardware specifics influence execution risks. This approach would be particularly valuable for database

systems deployed across heterogeneous hardware environments or for systems that must adapt to hardware upgrades without complete retraining.

- **Distributed Processing and Scalability:** Extending the proposed risk-aware optimization approach to distributed query processing environments represents a critical next step for scaling to larger schemas and more complex workloads. While this thesis focused on demonstrating the feasibility and robustness of the approach on single-node query optimization, the experimental evaluation included queries with up to 27 joins, demonstrating applicability to fairly complex workloads. However, scaling to much larger schemas and distributed query processing environments, where the optimizer could account for parallelism, memory hierarchies, and network characteristics, is an important research direction. Future work in this area would involve extending the risk quantification framework to model distributed execution risks, including network latency uncertainties, data skew across nodes, and coordination overhead. The Graph Neural Network architecture described in Chapter 5 would need enhancement to capture distributed query characteristics, such as data partitioning strategies, join distribution patterns, and inter-node communication requirements. Additionally, the suboptimality risk formulation would require extension to account for the complex interactions between local and global optimization decisions in distributed environments. This research direction would enable robust query optimization in modern

distributed database systems, cloud-native architectures, and large-scale data processing platforms where traditional single-node approaches become insufficient.

8.7 Concluding Remarks

This thesis has introduced a principled approach to robust query optimization based on approximate probabilistic machine learning. By formalizing the concept of robustness, quantifying different types of risks, and developing techniques that mitigate these risks, we have addressed a critical gap in the field of database systems. Our experimental results demonstrate that risk-aware optimization strategies can significantly improve performance stability while maintaining or enhancing average execution times.

The journey from theoretical formulation to practical implementation, guided by the systematic DSR framework, has yielded valuable insights into the interplay between machine learning and database systems. It has shown that sophisticated neural architectures, when designed with domain-specific considerations, can effectively capture the complex relationships that influence query performance.

As database systems continue to evolve in response to growing data volumes, increasing query complexity, and diverse deployment environments, the need for robust optimization techniques will only intensify. The framework presented in this thesis lays a foundation

for addressing these challenges, offering both immediate practical benefits and long-term research directions. By embracing uncertainty rather than ignoring it, query optimizers can make more informed decisions that lead to more predictable, reliable, and efficient database performance.

In conclusion, this research represents a significant step toward realizing the vision of self-tuning database systems that can adapt to changing conditions while maintaining stable performance. As machine learning techniques continue to mature and integrate more deeply with database management systems, the principles of risk-aware optimization presented here will serve as valuable guideposts for future innovations in this domain.

References

[1] Ray tune: Hyperparameter tuning — ray 2.8.0, 2023.

[2] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U. Rajendra Acharya, Vladimir Makarenkov, and Saeid Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.

[3] Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of Data*, pages 181–192, 1999.

[4] Ahmad K. Z. Al Saedi, Rozaida Ghazali, and Mustafa Mat Deris. An efficient multi join query optimization for dbms using swarm intelligent approach. In *2014 Fourth*

World Congress on Information and Communication Technologies (WICT), pages 113–117, 2014.

[5] Ahmad K. Z. AlSaedi, Rozaida Ghazali, and Mustafa M. Deris. *An Efficient Multi Join Query Optimization for Relational Database Management System Using Two Phase Artificial Bee Colony Algorithm*, pages 213–226. Springer International Publishing, 2015.

[6] Brian Babcock and Surajit Chaudhuri. Towards a robust query optimizer: A principled and practical approach. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of Data*, pages 119–130, 2005.

[7] Shivnath Babu, Pedro Bizarro, and David DeWitt. Proactive re-optimization. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 107–118, 2005.

[8] Gerardo Beni and Jing Wang. *Swarm Intelligence in Cellular Robotic Systems*, pages 703–712. Springer, 1993.

[9] Kevin P. Bennett, Michael C. Ferris, and Yannis E. Ioannidis. A genetic algorithm for database query optimization. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1991.

[10] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01621459.2017.1285773.

[11] Björn Blohsfeld, Dieter Korus, and Bernhard Seeger. A comparison of selectivity estimators for range queries on metric attributes. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, page 239–250, New York, NY, USA, 1999. Association for Computing Machinery.

[12] Renata Borovica-Gajic, Stratos Idreos, Anastasia Ailamaki, Marcin Zukowski, and Campbell Fraser. Smooth scan: robust access path selection without cardinality estimation. *The VLDB Journal*, 27(4):521–545, 2018.

[13] Nicolas Bruno and Surajit Chaudhuri. Exploiting statistics on query expressions for optimization. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of Data*, pages 263–274, 2002.

[14] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Stholes: A multidimensional workload-aware histogram. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, page 211–222, New York, NY, USA, 2001. Association for Computing Machinery.

[15] Donald D. Chamberlin and Raymond F. Boyce. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, pages 249–264, 1974.

[16] Baoming Chang, Amin Kamali, and Verena Kantere. A novel technique for query plan representation based on graph neural nets. In Robert Wrembel, Silvia Chiusano, Gabriele Kotsis, A. Min Tjoa, and Ismail Khalil, editors, *Big Data Analytics and Knowledge Discovery*, pages 299–314, Cham, 2024. Springer Nature Switzerland.

[17] Baoming Chang, Amin Kamali, and Verena Kantere. Reqo: A robust and explainable query optimization cost model. *arXiv preprint arXiv:2501.17414*, 2025.

[18] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Random sampling for histogram construction: how much is enough? In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 436–447. ACM, 1998.

[19] Chungmin Melvin Chen and Nick Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 161–172, 1994.

[20] Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. LOGER: A learned optimizer towards generating efficient and robust query execution plans. *Proceedings of the VLDB Endowment*, 16(7):1777–1789, 2023.

[21] Xu Chen, Haitian Chen, Zibo Liang, Shuncheng Liu, Jinghong Wang, Kai Zeng, Han Su, and Kai Zheng. Leon: A new framework for ml-aided query optimization. *Proc. VLDB Endow.*, 16(9):2261–2273, May 2023.

[22] Yimeng Chen, Devin Jia, Ying Lei, and Bo Li. Ensemble masked autoencoder and gaussian mixture models for cardinality estimation. In *2023 5th International Conference on Data-driven Optimization of Complex Systems (DOCS)*, pages 1–7, 2023.

[23] S. Christodoulakis. Implications of certain assumptions in database performance evauation. *ACM Transactions on Database Systems*, 9(2):163–186, 1984.

[24] Francis Chu, Joseph Halpern, and Johannes Gehrke. Least expected cost query optimization: what can we expect? In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 293–302. Association for Computing Machinery, 2002.

[25] Francis Chu, Joseph Y. Halpern, and Praveen Seshadri. Least expected cost query optimization: an exercise in utility. In *Proceedings of the eighteenth ACM SIGMOD-*

SIGACT-SIGART symposium on Principles of database systems, PODS '99, pages 138–147. Association for Computing Machinery, 1999.

[26] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[27] David L. Poole. *Computational intelligence*. Oxford University Press, 1998.

[28] Amol Deshpande, Minos Garofalakis, and Rajeev Rastogi. Independence is good: dependency-based histogram synopses for high-dimensional data. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, page 199–210, New York, NY, USA, 2001. Association for Computing Machinery.

[29] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek Narasayya. Dsb: a decision support benchmark for workload-driven and traditional database systems. *Proc. VLDB Endow.*, 14(13):3376–3388, September 2021.

[30] Lyric Doshi, Vincent Zhuang, Gaurav Jain, Ryan Marcus, Haoyu Huang, Deniz Altinbüken, Eugene Brevdo, and Campbell Fraser. Kepler: Robust learning for parametric query optimization. *Proceedings of the ACM on Management of Data*, 1(1):109:1–109:25, 2023.

[31] A. Dutt and J.R. Haritsa. Plan bouquets: Query processing without selectivity estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1039–1050. ACM, 2014.

[32] Anshuman Dutt and Jayant R. Haritsa. Plan bouquets: A fragrant approach to robust query processing. *ACM Transactions on Database Systems*, 41(2):11:1–11:37, 2016.

[33] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. Selectivity Estimation for Range Predicates Using Lightweight Models. *Proceedings of the VLDB Endowment*, 12(9):1044–1057, 2019.

[34] Tansel Dökeroğlu and Ahmet Coşar. *Dynamic Programming with Ant Colony Optimization Metaheuristic for Optimization of Distributed Database Queries*, pages 107–113. Springer, 2012.

[35] Shuhuan Fan, Mengshu Hou, Rui Xi, and Wenwen Ma. Precision meets resilience: Cross-database generalization with uncertainty quantification for robust cost estimation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 581–590, 2024.

[36] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1905.06881*, 2019.

[37] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, September 1985.

[38] Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference, 2016.

[39] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1050–1059. PMLR, 2016. ISSN: 1938-7228.

[40] Jakob et al Gawlikowski. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, 56(1):1513–1589, 2023.

[41] Chandrika Giannella, Jiawei Han, Jian Pei, Xifeng Yan, and Philip S. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 191–200. SIAM, 2003.

[42] Leila Golshanara, S. M. T. Rajaee Rankoohi, and Hamed Shah-Hosseini. A multi-colony ant algorithm for optimizing join queries in distributed database systems. *Knowledge and Information Systems*, 39(1):175–206, 2014.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[44] G. Graefe, H. Kuno, and J.L. Wiener. Visualizing the robustness of query execution. In *CIDR 2009 - 4th Biennial Conference on Innovative Data Systems Research*. CIDR, 2009.

[45] Goetz Graefe. New algorithms for join and grouping operations. *Computer Science-Research and Development*, 27:3–27, 2012.

[46] Shirley Gregor and Alan R. Hevner. Positioning and presenting design science research for maximum impact. *MIS Quarterly*, 37(2):337–355, 2013.

[47] Dimitrios Gunopulos, George Kollios, Vassilis J Tsotras, and Carlotta Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *the VLDB Journal*, 14:137–154, 2005.

[48] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, page 311–322, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[49] Yuxing Han, Haoyu Wang, Lixiang Chen, Yifeng Dong, Xing Chen, Benquan Yu, Chengcheng Yang, and Weining Qian. Bytecard: Enhancing bytedance's data warehouse with learned cardinality estimation. In *Proceedings of the 2024 International Conference on Management of Data (SIGMOD '24)*, pages 41–54. ACM, 2024.

[50] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. Cardinality estimation in dbms: a comprehensive benchmark evaluation. *Proc. VLDB Endow.*, 15(4):752–765, December 2021.

[51] D. Harish, P.N. Darera, and J.R. Haritsa. Identifying robust plans through plan diagram reduction. *Proceedings of the VLDB Endowment*, 1(1):1124–1140, 2008.

[52] Jayant R. Haritsa. Robust query processing: mission possible. *Proceedings of the VLDB Endowment*, 13(12):3425–3428, 2020.

[53] Max Heimel, Martin Kiefer, and Volker Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1477–1492. Association for Computing Machinery, 2015.

[54] Axel Hertzschuch, Guido Moerkotte, Wolfgang Lehner, Norman May, Florian Wolf, and Lars Fricke. Small selectivities matter: Lifting the burden of empty samples. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, pages 697–709. Association for Computing Machinery, 2021.

[55] Axel Hertzschuch, Matthias Uflacker, and Thomas Zeitz. Small selectivities matter: Lifting the burden of empty samples. In *Proceedings of the 43rd International Con-*

ference on Very Large Data Bases (VLDB), pages 2456–2468. VLDB Endowment, 2020.

[56] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[57] Benjamin Hilprecht and Carsten Binnig. Zero-shot cost models for out-of-the-box learned cost prediction. *Proc. VLDB Endow.*, 15(11):2361–2374, July 2022.

[58] Benjamin Hilprecht, Samuel Drews, and Carsten Binnig. DeepDB: Learn from data, not from queries! *Proceedings of the VLDB Endowment*, 13(7):992–1005, 2020.

[59] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021.

[60] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, page 647–658, New York, NY, USA, 2004. Association for Computing Machinery.

[61] Yannis Ioannidis and Stavros Christodoulakis. On the propagation of errors in the size of join results. In *ACM SIGMOD Record*, volume 20, pages 268–277. ACM, 1991.

[62] Yannis E. Ioannidis. The history of histograms (abridged version). In *Proceedings of the 2003 VLDB conference*, pages 19–30, 2003.

[63] Mohammad Jafarinejad and Mostafa Amini. Multi-join query optimization in bucket-based encrypted databases using an enhanced ant colony optimization algorithm. *Distributed and Parallel Databases*, 36(2):399–441, 2018.

[64] G. Jintao, L. Zhanhuai, and S. Jian. Thorough data pruning for join query in database system. *IEEE Transactions on Sustainable Computing*, pages 1–14, 2023.

[65] A.R. Kader, P. Boncz, S. Manegold, and M. Van Keulen. ROX: Run-time optimization of XQueries. In *SIGMOD-PODS'09 - Proceedings of the International Conference on Management of Data and 28th Symposium on Principles of Database Systems*, pages 615–626. ACM, 2009.

[66] Amin Kamali, Verena Kantere, and Calisto Zuzarte. Robust query optimization in the era of machine learning: State-of-the-art and future directions. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 5371–5375, 2024.

[67] Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. Robopt: A tool for robust workload optimization based on uncertainty-aware machine learning. In *Companion of the 2024 International Conference on Management of Data*, SIGMOD '24, page 468–471, New York, NY, USA, 2024. Association for Computing Machinery.

[68] Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. Roq: Robust query optimization based on a risk-aware learned cost model, 2024.

[69] Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. Robust plan evaluation based on approximate probabilistic machine learning. *Proc. VLDB Endow.*, 18(8):2626–2638, September 2025.

[70] Amin Kamali, Jarek Szlichta, Calisto Zuzarte, Yueting Chen, Verena Kantere, and Shaikh Quader. Ml4dm: The second workshop on the emerging applications of machine learning in modern data management. In *Proceedings of the 32nd Annual International Conference on Computer Science and Software Engineering*, CASCON '22, page 229–231, USA, 2022. IBM Corp.

[71] Amin Kamali, Calisto Zuzarte, and Verena Kantere. Emerging applications of machine learning in modern data management. In *Proceedings of the 31st Annual Inter-*

national Conference on Computer Science and Software Engineering, CASCON '21, page 284–286, USA, 2021. IBM Corp.

[72] Amin Kamali, Calisto Zuzarte, Verena Kantere, Jaroslaw Szlichta, Yueting Chen, Xiaohui Yu, and Ning Wang. Ml4dm '23: The third workshop on the emerging applications of machine learning in modern data management. In *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering*, CASCON '23, page 251–253, USA, 2023. IBM Corp.

[73] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Bertty Contreras-Rojas, Rodrigo Pardo-Meza, Anis Troudi, and Sanjay Chawla. Ml-based cross-platform query optimization. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1489–1500. IEEE, 2020.

[74] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[75] S. Karthik. Robust query processing. In *2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW)*, pages 226–230. IEEE, 2016.

[76] Srinivas Karthik, Jayant R. Haritsa, Sreyash Kenkre, and Vinayaka Pandit. A concave path to low-overhead robust query processing. *Proceedings of the VLDB Endowment*, 11(13):2183–2195, 2018.

[77] Srinivas Karthik, Jayant R. Haritsa, Sreyash Kenkre, Vinayaka Pandit, and Lohit Krishnan. Platform-independent robust query processing. *IEEE Trans. on Knowl. and Data Eng.*, 31(1):17–31, January 2019.

[78] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *International Conference on Neural Information Processing Systems*, pages 5580–5590, 2017.

[79] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019*, 2019.

[80] Andreas Kipf, Dimitri Vorona, Jonas Müller, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, Thomas Neumann, and Alfons Kemper. Estimating cardinalities with deep sketches. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD 2019*, pages 1937–1940. Association for Computing Machinery, 2019.

[81] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. Learning to optimize join queries with deep reinforcement learning, 2019.

[82] T. V. Vijay Kumar, Vijay Singh, and Ajay Kumar Verma. Distributed query processing plans generation using genetic algorithm. Master's thesis, National Institute of Technology Kurukshetra, 2011.

[83] Suyong Kwon, Woohwan Jung, and Kyuseok Shim. Cardinality estimation of approximate substring queries using deep learning. *Proc. VLDB Endow.*, 15(11):3145–3157, July 2022.

[84] M. Seetha Lakshmi and Shaoyu Zhou. Selectivity estimation in extensible databases - a neural network approach. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, page 623–627, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[85] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[86] Ju-Hong Lee, Deok-Hwan Kim, and Chin-Wan Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *Proceedings of the 1999 ACM*

SIGMOD International Conference on Management of Data, SIGMOD '99, page 205–214, New York, NY, USA, 1999. Association for Computing Machinery.

[87] Sangoh Lee, Kyoungmin Kim, and Wook-Shin Han. ASM in action: Fast and practical learned cardinality estimation. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24)*, pages 460–463. ACM, 2024.

[88] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, nov 2015.

[89] Beibin Li, Yao Lu, and Srikanth Kandula. Warper: Efficiently adapting learned cardinality estimators to data and workload drifts. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1920–1933, New York, NY, USA, 2022. Association for Computing Machinery.

[90] Jiexing Li, Arnd Christian König, Vivek Narasayya, and Surajit Chaudhuri. Robust estimation of resource consumption for sql queries using statistical techniques. *Proc. VLDB Endow.*, 5(11):1555–1566, July 2012.

[91] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning, 2020.

[92] Ning Li, Yu Liu, Ying Dong, and Jun Gu. *Application of Ant Colony Optimization Algorithm to Multi-Join Query Optimization*, pages 189–197. Springer, 2008.

[93] Pengfei Li, Wenqing Wei, Rong Zhu, Bolin Ding, Jingren Zhou, and Hua Lu. Alece: An attention-based learned cardinality estimator for spj queries on dynamic workloads. *Proc. VLDB Endow.*, 17(2):197–210, October 2023.

[94] Chang Liu, Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. A novel framework for join order selection based on reinforcement and representation learning. In *2024 IEEE International Conference on Big Data (BigData)*, pages 8757–8761, 2024.

[95] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. Cardinality estimation using neural networks. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*, CASCON '15, page 53–59, USA, 2015. IBM Corp.

[96] Jeremiah Zhe Liu, Shreyas Padhy, Jie Ren, Zi Lin, Yeming Wen, Ghassen Jerfel, Zachary Nado, Jasper Snoek, Dustin Tran, and Balaji Lakshminarayanan. A simple

approach to improve single-model deep uncertainty via distance-awareness. *Journal of Machine Learning Research*, 24(42):1–63, 2023.

[97] Jie Liu, Wenqian Dong, Qingqing Zhou, and Dong Li. Fauce: fast and accurate deep ensembles with uncertainty for cardinality estimation. *Proc. VLDB Endow.*, 14(11):1950–1963, July 2021.

[98] Hongjun Lu and Rudy Setiono. Effective query size estimation using neural networks. *Applied Intelligence*, 16(3):173–183, 2002.

[99] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, 1995.

[100] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, pages 1275–1288. Association for Computing Machinery, 2024.

[101] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: a learned query optimizer. *Proc. of VLDB Endow.*, 12(11):1705–1718, 2019.

[102] Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM'18, New York, NY, USA, 2018. Association for Computing Machinery.

[103] Ryan Marcus and Olga Papaemmanouil. Plan-structured deep neural network models for query performance prediction. *Proc. VLDB Endow.*, 12(11):1733–1746, 2019.

[104] Volker Markl, Vijayshankar Raman, David Simmen, Guy Lohman, Hamid Pirahesh, and Miso Cilimdzic. Robust query processing through progressive optimization. In *ACM SIGMOD*, pages 659–670, 2004.

[105] Zizhong Meng, Peizhi Wu, Gao Cong, Rong Zhu, and Shuai Ma. Unsupervised selectivity estimation by integrating gaussian mixture models and an autoregressive model. In *EDBT*, pages 2–247, 2022.

[106] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[107] Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, 1997. OCLC: 36417892.

[108] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment*, 2(1):982–993, 2009.

[109] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 1287–1293. AAAI Press, 2016.

[110] Chiraz Moumen, Franck Morvan, and Abdelkader Hameurlain. Handling estimation inaccuracy in query optimization. In *Web Technologies and Applications*, pages 355–367, 2016.

[111] Magnus Müller, Lucas Woltmann, and Wolfgang Lehner. Enhanced featurization of queries with mixed combinations of predicates for ML-based cardinality estimation. In *Proceedings of the 26th International Conference on Extending Database Technology (EDBT)*, pages 273–284, 2023.

[112] Raghunath Othayoth Nambiar and Meikel Poess. The making of TPC-DS. In *VLDB*, volume 6, pages 1049–1058, 2006.

[113] Parimarjan Negi, Matteo Interlandi, Ryan Marcus, Mohammad Alizadeh, Tim Kraska, Marc Friedman, and Alekh Jindal. Steering query optimizers: A practi-

cal take on big data workloads. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2557–2569, New York, NY, USA, 2021. Association for Computing Machinery.

[114] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. Flow-loss: Learning cardinality estimates that matter. *Proc. VLDB Endow.*, 14(11):2019–2032, jul 2021.

[115] Parimarjan Negi, Ziniu Wu, Andreas Kipf, Nesime Tatbul, Ryan Marcus, Sam Madden, Tim Kraska, and Mohammad Alizadeh. Robust query driven cardinality estimation under changing workloads. In *Proceedings of the VLDB Endowment*, volume 16, pages 1520–1533, 2023.

[116] Parimarjan Negi, Ziniu Wu, Arash Nasr-Esfahany, Harsha Sharma, Mohammad Alizadeh, Tim Kraska, and Sam Madden. Os pre-trained transformer: Predicting query latencies across changing system contexts, 2024.

[117] Pedro Neuhaus, Jorge Couto, Jonas Wehrmann, Diego Ruiz, and Felipe Meneguzzi. Gadis: A genetic algorithm for database index selection. In *The 31st International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 39–42, 2019.

[118] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3), March 2018.

[119] D.A. Nix and A.S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pages 55–60 vol.1, 1994.

[120] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, 2019.

[121] Mayur Patil and Amr Magdy. LATEST: Learning-assisted selectivity estimation over spatio-textual streams. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1607–1618, 2021.

[122] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[123] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings*

of the 1996 ACM SIGMOD international conference on Management of Data, pages 294–305. ACM, 1996.

[124] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. Selectivity estimation without the independence assumption. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, pages 486–495, 1997.

[125] V. Raman, V. Markl, D. Simmen, G. Lohman, and H. Pirahesh. Progressive optimization in action. In *Proceedings 2004 VLDB Conference: The 30th International Conference on Very Large Databases (VLDB)*, pages 1337–1340. 2004.

[126] V. Raman, L. Qiao, W. Han, I. Narang, Y.-L. Chen, K.-H. Yang, and F.-L. Ling. Lazy, adaptive rid-list intersection, and its application to index anding. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 773–784. ACM, 2007.

[127] Lukas Reiner, Andreas Kipf, Thomas Kipf, Bastian Radke, Viktor Leis, Peter Boncz, Alfons Kemper, and Thomas Neumann. Sample-efficient cardinality estimation using geometric deep learning. *Proceedings of the VLDB Endowment*, 17(5):740–753, 2024.

[128] Silvan Reiner and Michael Grossniklaus. Sample-efficient cardinality estimation using geometric deep learning. *Proceedings of the VLDB Endowment*, 17(4):740–752, 2023.

[129] B. Răducanu, P. Boncz, and M. Zukowski. Micro adaptivity in vectorwise. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1231–1242. ACM, 2013.

[130] Wolfgang Scheufele and Guido Moerkotte. Efficient dynamic programming algorithms for ordering expensive joins and selections. In *Advances in Database Technology — EDBT'98*, volume 1377 of *Lecture Notes in Computer Science*, pages 199–215. Springer, 1998.

[131] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, SIGMOD '79, page 23–34, New York, NY, USA, 1979. Association for Computing Machinery.

[132] K. D. Seppi, J. W. Barnes, and C. N. Morris. A bayesian approach to database query optimization. *ORSA Journal on Computing*, 5(4):410–419, 1993.

[133] D. Sharkova, A. Chernokoz, A. Trofimov, N. Sokolov, E. Gorshkova, I. Kuralenok, and B. Novikov. Adaptive SQL query optimization in distributed stream processing: A preliminary study. In *Communications in Computer and Information Science*, volume 1457 CCIS, pages 96–109. Springer, 2022.

[134] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2021.

[135] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. Cost models for big data query processing: Learning, retrofitting, and our findings. In *ACM SIGMOD*, pages 99–113, 2020.

[136] Wiktor Stadnik and Ziemowit Nowak. The impact of web pages' load time on the conversion rate of an e-commerce platform. In Ngoc Thanh Nguyen, Ryszard Kowalczyk, Jorge Casillas, Yannis Manolopoulos, Lazaros S. Iliadis, and Bogdan Trawiński, editors, *Computational Collective Intelligence - 9th International Conference, ICCCI 2017, Nicosia, Cyprus, September 27-29, 2017, Proceedings*, volume 10448 of *Lecture Notes in Computer Science*, pages 432–441. Springer, 2017.

[137] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. LEO - DB2's learning optimizer. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*, pages 19–28. Morgan Kaufmann, 2001.

[138] Ji Sun and Guoliang Li. An end-to-end learning-based cost estimator. *Proc. VLDB Endow.*, 13(3):307–319, November 2019.

[139] Jarek Szlichta, Calisto Zuzarte, Verena Kantere, Amin Kamali, Andrew Chai, Xiaohui Yu, Chang Liu, and Baoming Chang. Ml4dm '24: The fourth workshop on the emerging applications of machine learning in modern data management. 2024.

[140] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks, 2015.

[141] Saravanan Thirumuruganathan, Suraj Shetiya, Nick Koudas, and Gautam Das. Prediction intervals for learned cardinality estimation: an experimental evaluation. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 3051–3064. IEEE, 2022.

[142] Pooja Tiwari and S. V. Chande. *Optimal Ant and Join Cardinality for Distributed Query Optimization Using Ant Colony Optimization Algorithm*, pages 385–392. Springer, 2019.

[143] I. Trummer, S. Moseley, D. Maram, S. Jo, and J. Antonakakis. SkinnerDB: Regret-bounded query evaluation via reinforcement learning. In *Proceedings of the VLDB Endowment*, volume 11, pages 2074–2077. VLDB Endowment, 2018. Issue: 12.

[144] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal*, 22(1):3–27, February 2013.

[145] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[146] John Venable, Jan Pries-Heje, and Richard Baskerville. Feds: A framework for evaluation in design science research. *European Journal of Information Systems*, 25(1):77–89, 2014.

[147] Fang Wang, Xiao Yan, Man Lung Yiu, Shuai Li, Zunyao Mao, and Bo Tang. Speeding up end-to-end query execution via learning-based progressive cardinality estimation. *Proceedings of the ACM on Management of Data*, 1:1–25, 2023.

[148] Jiayi Wang, Chengliang Chai, Jiabin Liu, and Guoliang Li. FACE: A normalizing flow based cardinality estimator. *Proceedings of the VLDB Endowment*, 15(1):72–84, 2021.

[149] Ning Wang, Amin Kamali, Verena Kantere, Calisto Zuzate, Vincent Corvinelli, Brandon Frendo, and Steve Donoghue. A hybrid cost model for evaluating query execution plans. In *2023 IEEE Sixth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 133–138, 2023. ISSN: 2831-7203.

[150] Z. Wang, Q. Zeng, N. Wang, H. Lu, and Y. Zhang. CEDA: Learned cardinality estimation with domain adaptation. In *Proceedings of the VLDB Endowment*, volume 16, pages 3934–3937. VLDB Endowment, 2023. Issue: 12.

[151] Z. Wei and I. Trummer. SkinnerMT: Parallelizing for efficiency and robustness in adaptive query processing on multicore platforms. *Proceedings of the VLDB Endowment*, 16(4):905–917, 2022.

[152] Lianggui Weng, Rong Zhu, Di Wu, Bolin Ding, Bolong Zheng, and Jingren Zhou. Eraser: Eliminating performance regression on learned query optimizer. *Proc. VLDB Endow.*, 17(5):926–938, May 2024.

[153] Michael Wiegand. Site speed is (still) impacting your conversion rate, 2022. Accessed: 2025-01-13.

[154] Florian Wolf, Michael Brendle, Norman May, Paul R. Willems, Kai-Uwe Sattler, and Michael Grossniklaus. Robustness metrics for relational query execution plans. *Proceedings of the VLDB Endowment*, 11(11):1360–1372, 2018.

[155] Florian Wolf, Norman May, Paul R. Willems, and Kai-Uwe Sattler. On the calculation of optimality ranges for relational query execution plans. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 663–675. Association for Computing Machinery, 2018.

[156] Chenggang Wu, Alekh Jindal, Saeed Amizadeh, Hiren Patel, Wangchao Le, Shi Qiao, and Sriram Rao. Towards a learning optimizer for shared clouds. *Proceedings of the VLDB Endowment*, 12(3):210–222, 2018.

[157] Peizhi Wu and Gao Cong. A unified deep model of learning from both data and queries for cardinality estimation. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, pages 2009–2022, 2021.

[158] Wentao Wu, Yu Xu, Chin-Yung Cher, Pradeep Seshadri, Hiren Patel, Junyi Wang, and Vivek R. Narasayya. Sampling-based query re-optimization. In *Proceedings of the 2016 International Conference on Management of Data*, pages 677–692, 2016.

[159] Ziniu Wu, Pei Yu, Peilun Yang, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. A unified transferable model for ml-enhanced dbms. *arXiv preprint arXiv:2105.02418*, 2021.

[160] Haibo Xiu, Pankaj K. Agarwal, and Jun Yang. Parqo: Penalty-aware robust plan selection in query optimization, 2024.

[161] Jiani Yang, Sai Wu, Dongxiang Zhang, Jian Dai, Feifei Li, and Gang Chen. Rethinking learned cost models: Why start from scratch? *Proc. ACM Manag. Data*, 1(4), December 2023.

[162] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. Balsa: Learning a query optimizer without expert demonstrations. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 931–944, New York, NY, USA, 2022. Association for Computing Machinery.

[163] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. NeuroCard: One cardinality estimator for all tables. *Proceedings of the VLDB Endowment*, 14(1):61–73, 2021.

[164] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. Deep unsupervised cardinality estimation. *Proceedings of the VLDB Endowment*, 13(3):279–292, 2019.

[165] Shaoyi Yin, Abdelkader Hameurlain, and Franck Morvan. Robust query optimization methods with respect to estimation errors: A survey. *ACM SIGMOD Record*, 44(3):25–36, 2015.

[166] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. Cost-based or learning-based? a hybrid query optimizer for query plan selection. *Proc. VLDB Endow.*, 15(13):3924–3936, September 2022.

[167] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. Cost-based or learning-based? a hybrid query optimizer for query plan selection. *Proceedings of the VLDB Endowment*, 15(13):3924–3936, 2022.

[168] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. Reinforcement learning with tree-lstm for join order selection. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1297–1308, 2020.

[169] Tianjing Zeng, Junwei Lan, Jiahong Ma, Wenqing Wei, Rong Zhu, Pengfei Li, Bolin Ding, Defu Lian, Zhewei Wei, and Jingren Zhou. Price: A pretrained model for cross-database cardinality estimation. *arXiv preprint arXiv:2406.01027*, 2024.

[170] Jintao Zhang, Chao Zhang, Guoliang Li, and Chengliang Chai. AutoCE: An accurate and efficient model advisor for learned cardinality estimation. In *Proceedings of the 39th IEEE International Conference on Data Engineering (ICDE)*, pages 2621–2633. IEEE, 2023.

[171] Jintao Zhang, Chao Zhang, Guoliang Li, and Chengliang Chai. Pace: Poisoning attacks on learned cardinality estimation. *Proc. ACM Manag. Data*, 2(1), March 2024.

[172] Wangda Zhang, Matteo Interlandi, Paul Mineiro, Shi Qiao, Nasim Ghazanfari, Karlen Lie, Marc Friedman, Rafah Hosn, Hiren Patel, and Alekh Jindal. Deploying

a steered query optimizer in production at microsoft. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2299–2311, 2022.

[173] Yingjun Zhang, Guoliang Li, Jian Li, Nan Tang, and Ji Sun. Lightweight and accurate cardinality estimation by neural network gaussian process for approximate complex event processing. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*, pages 2046–2059, 2022.

[174] Yunjia Zhang, Yannis Chronis, Jignesh M. Patel, and Theodoros Rekatsinas. Simple adaptive query processing vs. learned query optimizers: Observations and analysis. *Proceedings of the VLDB Endowment*, 16(11):2962–2975, 2023.

[175] Kangfei Zhao, Jeffrey Xu Yu, Zongyan He, and Hao Zhang. Uncertainty-aware cardinality estimation by neural network gaussian process. *arXiv preprint arXiv:2107.08706*, 2021.

[176] Wei Zheng, Xinyu Jin, Fang Deng, Shijun Mo, Yizhi Qu, Yong Yang, Xiaoyu Li, Siyu Long, Chen Zheng, Jie Liu, and Zhen Xie. Database query optimization based on parallel ant colony algorithm. In *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*, pages 653–656, 2018.

[177] Yihang Zheng, Chen Lin, Xian Lyu, Xuanhe Zhou, Guoliang Li, and Tianqing Wang. Automating localized learning for cardinality estimation based on xgboost. *Knowledge and Information Systems*, 2024.

[178] Jianqiao Zhu, Navneet Potti, Saket Saurabh, and Jignesh M. Patel. Looking ahead makes query plans robust: making the initial case with in-memory star schema data warehouse workloads. *Proceedings of the VLDB Endowment*, 10(8):889–900, 2017.

[179] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. Lero: A learning-to-rank query optimizer. *Proc. VLDB Endow.*, 16(6):1466–1479, February 2023.

[180] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. FLAT: Fast, lightweight and accurate method for cardinality estimation. *Proceedings of the VLDB Endowment*, 14(9):1489–1502, 2021.

[181] Sergey Zinchenko and Sergey Iazov. Hero: Hint-based efficient and reliable query optimizer. *arXiv preprint arXiv:2412.02372*, 2024.

APPENDICES

Appendix A Hint Sets Used for Plan Generation

Inspired by the finding reported by Bao [100], these hint sets were used which include the most influential among the total 46 hint sets: [disable nljn], [disable nljn, disable iscan], [disable hsjn], [disable hsjn, disable iscan], [disable mgjn], [disable mgjn, disable iscan], [disable nljn, disable mgjn], [disable nljn, disable mgjn, disable iscan], [disable nljn, disable hsjn], [disable nljn, disable hsjn, disable iscan], [disable mgjn, disable hsjn], [disable mgjn, disable hsjn, disable iscan]