A Robust and Explainable Query Optimization Cost Model Based on Bidirectional Graph Neural Networks

by

Baoming Chang

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
Master of Computer Science

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

Abstract

Learning representations for query plans play a pivotal role in machine learning-based query optimizers of database management systems. To this end, particular model architectures are proposed in the literature to transform the tree-structured query plans into representations with formats learnable by downstream machine learning models. However, existing research rarely compares and analyzes the query plan representation capabilities of these tree models and their direct impact on the performance of the overall optimizer. In this thesis, we conduct a comparative study of existing tree models and innovatively propose a tree model architecture based on Bidirectional Graph Neural Networks (Bi-GNN) aggregated by Gated Recurrent Units (GRUs) and experimentally show its significant improvement in the performance of cost estimation and plan selection tasks.

The primary objective of query optimizers in relational database management systems (RDBMSs) is to identify the optimal execution plan with the lowest estimated cost for a given query. However, the inherent uncertainty in data and model parameters often leads to inaccuracies in cost estimation, resulting in the selection of suboptimal execution plans and not sufficiently robust query performance. In this thesis, we propose a novel learning-to-rank cost model that effectively quantifies the uncertainty associated with query execution cost predictions. This model integrates the predicted costs and their uncertainties using a learning-based mechanism and adapts by analyzing pairwise comparisons of integrated values, enabling more precise and adaptive uncertainty quantification and integration. We provide experimental evidence that our model significantly enhances query optimization's robustness, surpassing the capabilities of existing state-of-the-art techniques.

In addition, the decision-making processes of machine learning models often present an inherent lack of transparency, making their predictions difficult for humans to understand or trust. This issue exists in learning-based query optimization cost models as well. To address this, we propose a novel explainability technique specifically designed for learning-based cost models. This is the first technique to incorporate explainability into the learning-based cost model by assessing the influence of subtrees in the query plan on the final cost prediction. It explains the impact of specific subgraphs or nodes by analyzing the inclusion relationships between subtrees. This technique can be seamlessly integrated into any learning-based cost model and be trained concurrently with it. Our experiments demonstrate that combining this technique can substantially improve the explainability and credibility of the cost model while further improving its cost estimation performance.

By incorporating these innovations, we propose a comprehensive cost model for a Robust and Explainable Query Optimizer, Reqo, which simultaneously improves the performance of three critical dimensions of query optimization: cost estimation accuracy, plan selection robustness, and explainability. Our experimental results demonstrate that each dimension surpasses the corresponding state-of-the-art cost models.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Verena Kantere, for her unwavering support throughout my graduate journey. Her generous financial and resource support, along with her constant guidance, has been instrumental in shaping both my research and personal development. Prof. Kantere has been an exceptional supervisor, offering not only academic insight but also moral support and encouragement that greatly boosted my confidence as a researcher. I am really thankful for the opportunity she provided by inviting me to be her student, which opened the door to academic research and allowed me to embark on an entirely new career path.

I extend my sincere thanks to Amin Kamili, a PhD candidate at the University of Ottawa, whose profound technical expertise played a crucial role in my research. Amin has always been patient in answering my technical or personal questions, and his guidance has been invaluable throughout my master's journey. His continuous support has impacted my professional development, for which I am truly grateful.

I am also deeply thankful to my parents for their financial support, which allowed me to pursue my education abroad. Their unwavering encouragement, trust, and belief in my potential have been my greatest sources of motivation. Their support has inspired me to overcome challenges and persevere in pursuing my goals.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Query optimization is a critical component of a database management system due to its difficulty and importance in query execution performance. The process of accurately and efficiently estimating the cost of generated candidate query execution plans and selecting the optimal plan is always a challenge in the field of query optimization. A query execution plan is typically represented as a tree, where the nodes contain information about operators used to access, join, or aggregate data, and the edges contain dependencies between the parent and child nodes. An optimal plan tree enables the database management system to access and manipulate data efficiently. Traditional query optimizers employ cost models to estimate the amount of processing data using plan trees. These cost models rely heavily on statistical methods such as Histograms [56], which are susceptible to large errors because of their inability to capture characteristics such as join-crossing correlations effectively.

The development of machine learning provides a promising solution to improve query optimization and has been proposed in a wide range of applications in this field. Recent machine learning-based query optimizers employ a similar structure, as shown in Figure 1.1. In these frameworks, node information from a query plan tree is first encoded into node features. Then, a tree model transforms and aggregates these into graph-level representations for each candidate query plan, and a cost estimator predicts execution costs based on these representations, enabling the optimizer to select the most efficient plan. The challenge in this process is effectively transforming the tree-structured query plan into a graph-level vector while preserving as much of the original node features and structural information as possible. The quality of representation generated by the tree model is crucial to ensure the cost estimator model makes accurate predictions and allows the optimizer to make informed decisions on plan selection.

Current research in query optimization often lacks direct performance comparisons of the representation abilities of these tree models, focusing instead on the general performance of the whole query optimization process [175]. Therefore, in this thesis, we evaluate the learning capability of mainstream tree models in query plan representation under more

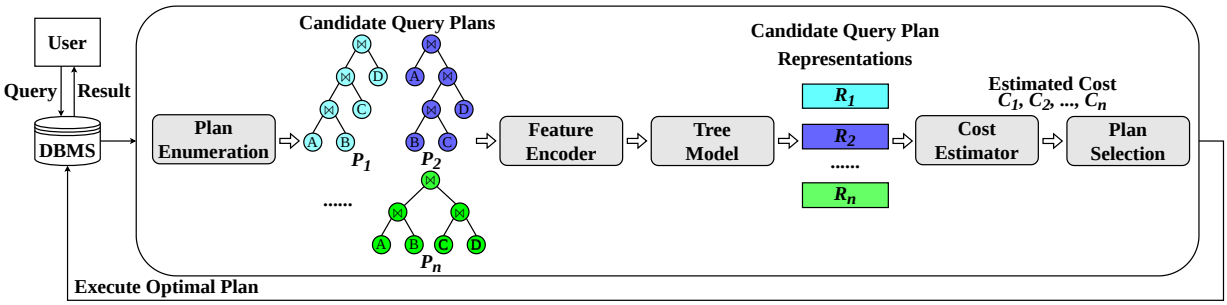


Figure 1.1: Machine learning-based query optimizer framework.

complex workloads, comparing and analyzing their performance in cost estimation tasks. In addition, unlike previous studies [175], instead of pairwise comparisons, we compare and analyze the tree model's plan selection capability in a way that is closer to actual usage scenarios by selecting an optimal plan from multiple candidate query plans to explore how different tree models affect the performance of plan selection.

Our research also explores the possibility of using Graph Neural Networks (GNNs) [177] as tree models. Although GNNs are being developed and have achieved success across various graph-based domains, their application within query plan cost estimation remains insufficiently explored. In this thesis, we explore the potential of employing GNNs for query execution plan representation learning. We also propose a novel tree model architecture for query plan representation based on bidirectional GNN [114] and a GRU-based Aggregation method [9], which can capture the intricacies of query plan trees more accurately and robustly, setting a new stage for query plan representation learning.

Furthermore, plan selection within a query optimizer relies on ranking the estimated costs of all candidate query plans by the cost model, which means the optimal plan selected by the model tends to have the lowest estimated cost. This methodology is effective only under the assumption that the estimation of the cost model is relatively accurate, but due to the complexity and uncertainty of the query plan and the database, it is unrealistic for the machine learning-based cost model to achieve in real-world scenarios. In this context, due to model estimation errors and non-conforming environments, the plan selected by the query optimizer may not be the optimal solution with the shortest actual runtime. Query optimization methods that are less sensitive to such estimation errors and do not rely on simplifying assumptions are considered robust [58], which is also one of the critical avenues for improving the overall performance of the optimizer.

Recent learning-based cost models have increasingly addressed robustness in query optimization. Some approaches [89, 88, 12] implement it by mitigating misestimation of the input parameters or minimizing the possibility of selecting a worse plan through specific mechanisms. However, the state-of-the-art approaches [173, 82, 15, 26, 58] have begun to quantify the uncertainty in data and model predictions, applying these measures to improve the robustness of query optimization. Among them, a new approach introduced by Roq [58] attempts to quantify uncertainties and risks based on approximate probabilistic ML and utilizes these quantifications through strategies to achieve more robust plan selection. Despite its superior robust performance over other machine learning strategies,

its model training part including cost and uncertainty estimation, and the plan selection strategy are independent, which makes the model unable to self-improve through feedback from plan selection results. Therefore, building upon Roq's work, we innovatively introduce a learnable parameter to integrate estimated cost and quantified uncertainty and adopt ranking loss on the integrated values in the form of query plan pairs to make plan comparison considered uncertainty a part of cost model training, which enables the model to automatically learn and optimize uncertainty estimates and integration strategy and makes the model has relatively accurate cost predictions while being more robust.

In addition, although our proposed tree model based on bidirectional GNN can effectively convert tree-structured query plan graphs into representations and outperform existing state-of-the-art tree models, due to the 'black box' nature of machine learning, the underlying rationales of the tree model outputs are still not straightforward for humans to understand and trust. This challenge is known as the problem of explainability for GNNs, which is also a limitation in current machine learning-based cost models. These approaches always overlook the explainability of the cost model, which means that the model theoretically aggregates plan node features and plan topology to generate cost predictions. Still, it remains unclear how specific subgraphs in the query plan significantly influence the cost model predictions. Existing state-of-art GNN explainability techniques [158, 85, 165, 166, 54, 14] mainly extract subgraphs through defined patterns and calculate the mutual information between the subgraphs and the original graph representation to find the most influential subgraph. However, due to the characteristics of the query plan tree, the existing explainers are not suitable for the field of query plan cost estimation, extracting the subgraphs randomly from the query plan for prediction will inevitably lead to the query semantics or operations containing key relations missing.

Therefore, in this study, we introduce explainability into query optimization learning-based cost models for the first time. We utilize the subtrees of the query plan as the subgraph extraction strategy to ensure the integrity of the query syntax and plan operations and avoid losing key nodes containing essential relations, such as leaf nodes. We develop an innovative explainability technique capable of automatically learning the correlation value between the representations of different subtrees within the query plan and the overall query plan representation as the contribution of the subtrees to final cost prediction during the cost model training process. The explainer model identifies the specific subgraph or operation that significantly influences cost estimation by calculating the contribution ratio of specific subgraphs through the inclusion relationships between subtrees. The technique markedly enhances the explainability of our cost model while leveraging the subtrees runtime generated by the RDBMS during the execution of the query plan as extra labels, thereby fully utilizing the workload information. This allows the model to learn and explain the impact of different subgraphs on the final prediction, further improving the model's representation ability and cost estimation accuracy.

1.2 Methodology

This thesis adopts a comprehensive mixed-method approach, utilizing both experimental designs and theoretical analyses to explore advanced query optimization techniques. The methodology is structured to ensure rigorous evaluation and practical relevance, enhancing the cost estimation accuracy, robustness and explainability of query optimization through innovative machine-learning approaches.

1.2.1 Data Collection and Preprocessing

To ensure the relevance and applicability of our findings, we evaluate them using data collected from several benchmarks, including TPC-DS [109], TPC-H [108], STATS [45], and JOB-light (IMDB) [70]. These benchmarks collectively encompass a broad spectrum of database systems scenarios from decision support systems to transaction processing and complex join queries typical of real-world applications. Each benchmark provides distinct query scenarios and data structures, enabling comprehensive testing of our proposed models under varied conditions. The diverse nature of these datasets allows for robust simulation of different query optimization scenarios, thereby aiding in the training and testing of our models with an extensive range of query structures and workload complexities.

1.2.2 Tree model Comparative Analysis

We conduct a thorough comparative analysis of various tree models, including the state-of-the-art tree models, GNN-based approaches and our proposed tree model BiGG. The models' performance in representation learning and cost estimation is evaluated under the pressure of TPC-DS's complex workload conditions. This process tests the models' accuracy in estimating costs and their ability to select the most efficient execution plan among multiple candidates. This evaluation focuses on the comparison and analysis of different tree model theories and the capability to handle the intricacies of real-world query optimization.

1.2.3 Proposed Techniques Integration and Evaluation

The various components of our work, including tree model BiGG, ranking-to-learn uncertainty-aware cost model, and explainability technique, are integrated into a comprehensive cost model framework Reqo. This framework is then trained and subjected to rigorous tests using the four benchmarks to validate its effectiveness in application scenarios. The final evaluation focuses on improvements in cost estimation accuracy, plan selection robustness, and model explainability compared to existing state-of-the-art learning-based cost models. This methodology ensures that our research is grounded in practical testing scenarios, enabling us to draw relevant conclusions about the efficacy and applicability of our proposed models in modern database environments.

1.3 Thesis Contributions

To summarize, our main contributions are:

- We conduct a comprehensive analysis of mainstream tree models and GNN models to assess their performance in cost estimation and plan selection tasks.
- We propose a novel query plan tree model BiGG based on bidirectional GNN and GRU, demonstrating its superior representational capabilities and performance compared to existing tree models.
- We design an uncertainty-aware cost model that employs a ranking-to-learn mechanism capable of adaptively integrating the estimated cost and quantified uncertainty, enhancing the model's robustness for query plan selection.
- We introduce explainability into the learning-based cost model for the first time by proposing a learning-based technique that explains the impact of query plan subgraphs on final cost predictions.
- We propose a comprehensive query optimization cost model, Reqo, that integrates our three proposed techniques, experimentally showing significant improvements in cost estimation accuracy, robustness and explainability over state-of-art cost models.

1.4 Thesis Structure

In the rest of the thesis:

Chapter 2 outlines related literature reviews along with their merits and shortcomings covering traditional optimizers, learning-based tree models for query plan representation, explainability techniques for Graph Neural Networks (GNNs), learning-based techniques in query optimizers, and the application of robustness technologies in query optimization.

Chapter 3 presents the problem statements of query plan representation, current tree models, robustness and explainability in query optimization.

Chapter 4 describes the theoretical framework of our proposed cost model, including the tree model BiGG, the ranking-to-learn uncertainty-aware cost model, and the explainability technique.

Chapter 5 introduces the architecture and implementation details of our proposed comprehensive cost model framework.

Chapter 6 presents the comparative study of existing and GNN-based tree models' performance on cost estimation and plan selection tasks, and evaluates the performance of our proposed cost model Reqo and other state-of-the-art learning-based cost models in terms of accuracy, robustness, and explainability.

Chapter 7 concludes both the advantages and disadvantages of the proposed framework in the thesis and discusses possible enhancements for our future work.

Chapter 2

Literature Review

In this chapter, we first review the techniques employed in traditional query optimizer cost models in Section 2.1. We then look back to state-of-the-art learning-based query plan representation methods focusing on tree models in Section 2.2.1. Additionally, we explore advanced GNN methods that have been or could be applied to query plan representation in Section 2.2.2 and the studies concerning the explainability of GNNs in Section 2.3. Next, we analyze the advanced machine learning-based query optimizer frameworks in Section 2.4. Finally, we discuss studies on the robustness of query optimization in Section 2.5. These sections collectively inspire the contributions of this thesis.

2.1 Traditional Query Optimizer Cost Model

Traditional query optimizer cost models can be categorized into two primary classes:

Traditional Cardinality Estimation. Cardinality in query optimization represents the total number of rows returned by a query plan or sub-part of a query plan. Thus, an accurate cardinality estimation can lead the optimizer to choose the query plan with a lower cost to improve the query performance. Traditional cardinality estimation can generally divided into three classes as follows:

1. *Histogram-based methods:* Histogram-based cardinality estimation [56] is a technique used to estimate the query results sizes by assessing the distribution of data, which segment the database values into bins of the same depth or width and maintain counts of records in each bin, and then using the involved bins to estimate the total cardinality of a query and aiding in the selection of efficient execution plans. Since this method is simple in principle and performs relatively well with lower inference overhead, it has been widely used in commercial relational database management systems for cardinality estimation. For instance, PostgreSQL [113] and DB2 [124] both employ histogram-based cardinality estimation methods in their DBMS. These systems typically assume that all attributes are mutually independent and use a one-dimensional (1-D) histogram to represent the probability for each attribute. By mul-

tiplying these individual probabilities, they estimate the probability of complex relationships or tables. Other examples of histogram routines include multi-dimensional histogram-based methods [23, 40, 41, 97, 111, 135] which identify subsets of correlated attributes and model them as multi-dimensional histograms. However, due to the simple principle and inflexible strategy, these histogram methods have limitations in accuracy, especially in representing data correlations and adapting to frequently changing data distributions. Additionally, correcting and self-tuning histograms with query feedbacks [8, 35, 184, 123] can dynamically adjust histogram-based cardinality estimates in databases to improve accuracy by incorporating actual query results but at the cost of introducing additional overhead and more complex design to ensure they do not become a bottleneck themselves.

2. *Sketching-based methods:* Sketching method is a probabilistic approach that efficiently approximates the number of distinct cardinality in a dataset by using hashing to create a concise, fixed-size summary of data. Sketching methods like FM [34], MinCount [39], LinearCount [143], LogLog [28], and HyperLogLog [33] primarily share the same basic idea of converting tuple values into bitmaps and counting either the continuous zeros or the number of hits for each position and then deducing the approximate count of distinct values. PessEst [10] is a recently proposed method that exhibits state-of-the-art performance in some aspects, which leverages randomized hashing and data sketching to tighten the bound for multi-join queries and never underestimates the cardinality. Meanwhile, it has been verified to perform well in real-world DBMS [103]. While efficient and scalable, the sketching-based methods come with the limitation of providing only approximate results, not exact counts. This can lead to inaccuracies, especially when precise data is crucial for decision-making or analysis. Additionally, sketching accuracy also depends on the quality of the hash functions used, where poor hash functions can result in unreliable estimates. These limitations make sketching less suitable for applications where precise cardinality is necessary, or data integrity is crucial.

3. *Sampling-based methods:* Existing sampling-based methods [71, 79, 148, 102, 47, 61, 72, 176] select a representative subset from a database and using the sample to estimate the total number of distinct values or the overall data distribution of the entire dataset. Compared to simple direct sampling, some query-driven kernel-based methods [47, 61] apply kernel density estimation (KDE) [119] to a selected sample of database data, using a smoothing kernel function to estimate the probability density function relevant to specific queries. This approach adjusts dynamically to the query characteristics, providing a flexible and potentially more accurate estimate of query result sizes, but it requires careful selection of kernel parameters and can be computationally intensive. The index-based method [71] utilizes existing database indexes to efficiently draw representative data samples, using index statistics and stratified sampling to provide accurate and scalable estimates of query result sizes. However, it depends on the quality and coverage of indexes, which limits their effectiveness in indexed columns and requires maintaining these indexes in a dynamic data environment. Furthermore, the random walk-based methods [72, 176] utilize a

stochastic process to make decisions on subsequent data points and navigate through a dataset, collecting samples based on predefined probability rules to estimate the total size of query results efficiently. The main challenge with random-walk-based methods requires correcting biases due to non-uniform sampling and ensuring the walk's efficiency and representativeness in varying data structures. Thus, sampling-based methods are practical for large databases where full scans are too costly or time-consuming. Estimations about the entire dataset can be made based on the properties observed from the sample, such as the frequency and distribution of values. Although this approach can significantly reduce computational overhead, it may have accuracy issues, especially when the sample can not represent the distribution of the entire database or queries are complex with multiple joins or filters, leading to the "0-tuple problem" in which no results are returned from the samples.

Traditional Cost Estimation. Instead of only focusing on the estimation of cardinality, traditional cost estimation in query optimization considers the cost of combining multiple factors like the cost of sequential page fetch, random page fetch, CPU processing per tuple and operational executions. However, the effectiveness of this method heavily relies on accurately tuning the weights assigned to each of these factors, which can be easily influenced by the database system and the specific queries. Several studies [147, 80, 70] have worked on cost model tuning and focused on refining these weights to enhance the performance of traditional cost predictions.

2.2 Learning-based Query Plan Representation

Query plan representation is an essential step in estimating query plan costs in machine learning-based query optimizers. Whether the subsequent models in the optimizer can accurately capture and learn from node features and structural information contained in the query plan depends largely on the effectiveness of the tree model used to convert the tree-structured query plan into a format that can be learned by other models. In this section, we first review the existing research on tree models. Then, since the query plan itself is also a graph, we also explore the existing advanced GNN technology and summarize the application of GNN technologies in query plan representations.

2.2.1 Tree Model

Query plan representation is mainly studied under one of the following categories: studies proposing novel tree models and their comparative analysis.

Tree models. RNN-based models like Long Short-Term Memory (LSTM) [51] and its variants are commonly used in query plan representation part of learning-based query optimization works like [125, 164]. These models achieve long-term information memory by using gates (input, output, and forget) that control the flow of information in and out of cells and learning what kind of information should be retained or discarded based

on specific data and tasks, which can make the tree models learn the dependencies and relationships between different nodes while aggregating node features. However, these methods have to convert tree structure plans into sequential nodes as input, inevitably leading to the loss of tree structure information. Some works apply the self-attention mechanism to tree models, such as Saturn [83], which aggregates LSTM's hidden layers based on attention as a representation of the query plan, or QueryFormer [174], which uses the Transformer [132] to encode the query plan. Although these models have indeed improved their ability to represent query plans through the attention mechanism, they still require converting the query plan into a sequence as input, which still causes the loss of structural information. Machine learning models are designed explicitly for tree structures, for instance, Tree-structured LSTM (Tree-LSTM) [126], which is the updated version of standard LSTM designed to process tree-structure data using child-sum or N-ary structures to enable the parent node to receive inputs from multiple child nodes and capture the dependencies and structural information between parent and child nodes. Tree Convolutional Neural Network (Tree-CNN) [95], initially used in query plan representation in Neo and then widely used in learning-based cost models, is a variation of conventional convolutional neural networks designed for tree-structured data. It can use multi-layer triangular sliders to update the information between each group of parent and child nodes to achieve message passing between different nodes and allow the node features to learn the information of its child nodes and structural information. These methods allow information transfer between child nodes and parent node so that the learned node features can also contain structural information. While these approaches improve upon non-tree models, they still do not demonstrate good results in solving the problem of information dilution and effective aggregation of node features discussed in Chapter 3.

Comparative study. A significant contribution to this field is a study by Yao Z. et al. [175]. The researchers conducted a detailed analysis comparing the performance of mainstream feature encoders and tree models under various scenarios. However, their study asserts that the tree model has no obvious impact on the overall performance of the optimizer, which is not confirmed by the findings of the current research. Their analysis also has a limitation: it lacks a comprehensive assessment of tree models in the plan selection task since their evaluation only relies on pairwise index selection [24], which does not adequately reflect the complexities in practical query optimization scenarios. Furthermore, they do not isolate the impact of the tree model architectures on this downstream task's performance.

2.2.2 Graph Neural Networks

A Graph Neural Network (GNN) is a type of deep learning model specifically designed to process data represented by graphs. It accepts graphs as both input and output, performing optimizable transformation on all attributes (nodes, edges, global context) in the graph while preserving the graph structure itself from changing [115]. This allows the GNN model to effectively capture the topological structure and node information as well as the dependencies between nodes through edges. Although the machine learning techniques based on RNN and developed for trees mentioned above could broadly be classified as GNNs, in this

thesis, the GNNs specifically refer to those machine learning technologies that aggregate features from adjacent nodes through a series of transformation or combination functions in a deep learning framework directly designed for processing graphs. Nowadays, due to GNNs' unique mechanism, they have played an important role in many application fields, such as social network analysis, recommendation systems, bioinformatics and chemistry. In this section, we discuss the mechanism of the important components (Message Passing Layer and Pooling Layer) of the current popular GNN models, as well as their applications in query plan representation.

Graph Neural Networks Message Passing Layers

Message Passing is an important concept in GNN, which realizes the exchange and collection of information between nodes in the graph. In order to facilitate the demonstration and comparison of the mechanisms of different GNN Message Passing Layers, here we assume a shared situation in this section:

Given a node set V in a graph, for each node $v \in V$, the initial embedding is given by the node's features:

$$h_v^{(0)} = x_v \tag{2.1}$$

$N(v)$ represents the neighbour nodes of node v in graph V. Subsequently, for each layer $k = 1, 2, \ldots, K$, $h_v^{(k)}$ represents the embedding of node v at layer k, $f^{(k)}$ is the function applied to node features at layer k and $W^{(k)}$ is the matrics with learnable parameters employed to the aggregated v's neighbour's embeddings at layer k.

The following are the mechanisms and related studies of several currently mainstream GNN message-passing layers:

1. *Graph Convolutional Networks (GCN):* Graph Convolutional Network [64] is a revolutionary type of neural network for processing graphs and is also considered the basis of existing graph neural network frameworks. It introduces convolution operations into node graph data for the first time and simplifies the application of convolutions into operations on nodes and their neighbour nodes in the graph. The basic mechanism of GCN is shown in Equation 2.2:

$$h_v^{(k)} = f^{(k)} \left(W^{(k)} \cdot \frac{\sum_{u \in N(v)} h_u^{(k-1)}}{|N(v)|} + B^{(k)} \cdot h_v^{(k-1)} \right) \tag{2.2}$$

   In classic GCN framework, the function $f^{(k)}$ is applied to two terms: the first term is the result of applying the weight matrix $W^{(k)}$ to the mean embeddings of v's neighbours from the previous layer $(k-1)$, and the second term is the transformation of node v's previous embedding by the weight matrix $B^{(k)}$. For each step k, the function $f^{(k)}$, matrics $W^{(k)}$ and $B^{(k)}$ are shared across all nodes.

   By doing so, GCNs can learn the essential topological and feature information, enabling the extraction of node embeddings that reflect both local and global graph

structures and capturing the complex relationships between nodes even if the graph data is irregular. Some studies have improved the basic GCN architecture, such as AM-GCN [139], which enhances GCN's ability to fuse graph topology and node features, making it better at semi-supervised node classification. Similarly, SoGCN [138] employs second-order polynomials to enhance the model's capacity to approximate high-order polynomial filters, thereby improving the representation capabilities of GCNs.

2. *Graph Attention Networks (GAT):* Graph Attention Network [133] is an upgraded version of the GCN framework and improves it with the self-attention mechanism [132]. Compared with the GCN model that treats all neighbouring nodes with given equal or static predefined weights, GAT can dynamically assign weights to neighbour nodes based on their features, allowing the model to aggregate node information in a more flexible way and providing a more powerful representation of the local or global graph, which is particularly useful in complex graph structures. The mechanism of basic GAT is shown in the Equation 2.3:

$$
\begin{aligned}
h_v^{(k)} &= f^{(k)}\left(W^{(k)} \cdot \left[\sum_{u \in N(v)} \alpha_{vu}^{(k-1)} h_u^{(k-1)} + \alpha_{vv}^{(k-1)} h_v^{(k-1)}\right]\right) \\
\alpha_{vu}^{(k)} &= \frac{A^{(k)}\left(h_v^{(k)}, h_u^{(k)}\right)}{\sum_{w \in N(v)} A^{(k)}\left(h_v^{(k)}, h_w^{(k)}\right)} \quad \text{for all } (u, v) \in E.
\end{aligned}
\tag{2.3}
$$

$\alpha_{vu}^{(k)}$ represents the attention coefficient between node v and node u at the k-th layer and $A^{(k)}$ is the pairwise attention mechanism at the k-th layer. Attention weights $\alpha_{vu}^{(k)}$ are generated by $A^{(k)}$ and normalized such that the sum over all neighbours of each node v is 1. E represents the set of edges in the graph.

From this equation, we can observe that the biggest difference between GAT and GCN is that it uses a weighted mean for the embedding of neighbour nodes instead of directly taking the mean. Although the introduction of the self-attention mechanism brings higher overhead, it also enables GAT to construct more complex dependencies between nodes. Thus, GAT is more suitable in task scenarios where relational information is important, such as social networks, knowledge graphs, and molecular structures. Recent research has further developed the basic GAT framework. For example, the original GAT may generate a form of static attention, where attention weights do not dynamically change based on different query nodes, which may limit its representation capabilities. Therefore, in order to improve this problem, GATv2 [6] proposes a more expressive attention mechanism by adjusting the internal order of operations within GAT. TransformerConv [121] introduces the Graph Transformer mechanism and masks a certain proportion of labels to improve the representation ability of the model, especially in semi-supervised classification tasks. And SuperGAT [62] improves the performance of GAT when processing noisy graph samples.

3. *Graph Sample and Aggregate (GraphSAGE):* Graph Sample and Aggregate [43] is an innovative method developed for learning large-scale graph data. Based on GCN, it updates node features by sampling fixed-size node neighbourhoods and aggregating their feature information, which significantly improves the computing efficiency of node embedding. Its mechanism is shown in Equation 2.4.

$$h_v^{(k)} = f^{(k)}\left(W^{(k)} \cdot \left[\mathrm{AGG}_{u \in N(v)}\left(\left\{h_u^{(k-1)}\right\}\right), h_v^{(k-1)}\right]\right) \tag{2.4}$$

The sampling technology significantly reduces the computational complexity of GraphSAGE when processing large-scale data, and improves the generalization ability of the model to a certain extent, making it more suitable for handling the induction problem of unseen nodes after model training. This framework also allows us to freely choose appropriate aggregation functions according to the type of tasks to aggregate the sampled neighbour node features, such as mean, pooling and LSTM. These characteristics of the model give it a more powerful performance in processing tasks such as node classification, link prediction, and node embedding generation in large graphs. Its ability to learn from both local and global graph structures without the need for the entire graph at once makes GraphSAGE a suitable model for handling real-world graph data. However, precisely because of these characteristics, an inappropriate sampling strategy selection is likely to negatively impact performance. At the same time, this strategy also makes GraphSAGE less efficient than other GNN methods in capturing graph-level representation.

4. *Graph Isomorphism Network (GIN):* Graph Isomorphism Network [154] is also a kind of powerful GNN network architecture, aiming to capture the permutation invariance of graph data so that it can effectively distinguish different graph structures. Its message-passing mechanism is shown in Equation 2.5.

$$h_v^{(k)} = f^{(k)}\left(\sum_{u \in N(v)} h_u^{(k-1)} + \left(1 + \epsilon^{(k)}\right) \cdot h_v^{(k-1)}\right) \tag{2.5}$$

where $\epsilon^{(k)}$ is a learnable parameter that can be used to weigh self-embeddings at the k-th iteration. This mechanism enables each layer of GIN to update node embeddings by aggregating the embeddings of neighbouring nodes and combining its own embedding through a learnable parameter ϵ. With the help of ϵ, using a simple but powerful aggregation function can give the GIN model an expressiveness similar to the Weisfeiler-Lehman (WL) Graph Isomorphism Test [120]. This aggregation function essentially sums the transformed feature vectors of neighbouring nodes and the node itself, combined with learnable parameters to capture local and global graph structure. A recent study [52] extends GIN so that it can learn edge features. GIN's ability to effectively learn and represent different graph topologies enables it to perform well in various tasks such as graph classification and node classification, making it a leading architecture for graph-structured data learning.

5. *GNN Message Passing Layers Comparison:* We compared the advantages and disadvantages of the four kinds of GNN Message Passing Layers mentioned above, as shown in Table 2.1.

Table 2.1: Advantages and disadvantages of GNN message passing layers.

Model	**Advantage**	**Disadvantage**
GCN	Efficiently utilizes the structure of the graph; Good performance on tasks like node classification and graph classification; Simplifies computations by using a localized first-order approximation of spectral graph convolutions.	Assumes that all neighbours contribute equally to the node representation, which may not always be true; Can suffer from over-smoothing when many layers are used, leading to indistinguishable node representations.
GAT	Uses attention mechanisms to weigh the importance of nodes' neighbours, enabling adaptive node feature aggregation; Can capture more complex, node-specific relational information; Offers potentially better interpretability through attention weights.	Has more expensive overhead than GCN due to the attention mechanism; The increased number of parameters can lead to overfitting on smaller graphs.
GraphSAGE	Does not require the full graph to be loaded into memory, making it scalable to large graphs; Uses a sampling approach to aggregate features from a node's local neighbourhood; Can generalize to unseen nodes by its inductive learning capability	Sampling strategy can affect performance if not carefully designed; May not capture global graph properties as effectively as other methods.
GIN	Excels in representation power, matching the Weisfeiler-Lehman (WL) graph isomorphism test; adept at discerning graph structures.	Demands meticulous parameter tuning to prevent overfitting and may not perform as efficiently on large-scale graphs compared to other GNN variants.

Graph Neural Networks Pooling Layers

Message passing layers can update attributes in the graph, yet to obtain a graph representation of graph-level tasks, a mechanism is required to aggregate the final node embeddings and pass them to another neural network for prediction, where exactly the pooling layers come into play. The general architecture of pooling layers is shown in Equation 2.6. Given a graph G, a neural network PREDICT_G:

$$h_G = \text{PREDICT}_G\left(\text{AGG}_{v \in G}\left(\{h_v\}\right)\right) \tag{2.6}$$

where h_G represents the prediction results based on the graph-level embedding obtained by aggregating the final features of its nodes h_v.

Therefore, the selection of aggregation function plays a very important role in the readout phase of graph neural networks. Many studies [44, 154, 21, 73, 127] demonstrate that the selection of aggregation function has a great impact on the representation ability and performance of GNN models. Basic global pooling methods have been proven to be simple but effective and powerful, where global mean pooling captures the distribution or proportion of graph elements, global max aggregation can find the most representative elements, and global add pooling facilitates learning of the structural graph properties [154].

Recent works have shown that using multiple aggregations and learnable aggregations potentially leads to significant performance improvements. For instance, [21, 127] proposes aggregation methods that can combine and stack the outputs of multiple underlying aggregations, [44, 9] apply Multilayer Perceptron (MLP) or Recurrent Neural Network (RNN) based models such as LSTM or GRU to do the node aggregation of GNN, which enable learning-based mechanisms in aggregation and achieve sufficient performance improvements. Some works [76, 69, 3, 9] introduce self-attention mechanisms into aggregate functions, which apply a neural network model to learn node scores, then aggregate nodes with the scores as weights, or keep only the nodes with the top scores, throwing away the rest and repeating until only one node is left. Sort vertices of the graph to get a fixed-size node-order invariant representation of the graph, and then apply any standard neural network architecture. In addition to these, there are some powerful techniques with interesting mechanisms, such as SortPool [168] sort vertices of the graph to get a fixed-size node-order invariant representation of the graph, and then apply any standard neural network architecture. DiffPool [159] learns to aggregate vertices into clusters, creating a simplified graph where clusters take the place of original nodes, then apply a GNN over the coarser graph and repeat until only one cluster is left.

Graph Neural Networks in Query Plan Representation

Recently, some works [12, 58] have attempted to apply GNNs in query plan representation. However, there is no GNN model designed explicitly for query plan tree-structured graphs yet, which results in the existing GNN models not being able to achieve good performance in the application scenario of query plan representation directly. Therefore, these relevant works use the GNNs only to capture the query's join relationship in order to assist the representation learning by other tree models rather than being directly used for learning query plan trees. These methods obviously do not make full use of the powerful representation performance of the GNN model. Therefore, how to apply GNN technology to query plan representation is a very potential research direction.

2.3 Explainability in Graph Neural Networks

In recent years, graph neural networks have been widely used in many fields in the real world, many of which require prediction results to be very accurate and trustworthy, for small errors may cause very serious consequences [57]. Due to the black-box nature of machine learning, people do not know how the model makes decisions. Therefore, various

methods have been developed to explain and support the prediction results of the learning-based GNN model so that the predictions of the model can be understood by humans and increase the credibility of the model. In the realm of graph neural networks, the complex data structure composition rules and key attributes contained in both nodes and edges leading to the evaluation and finding of the substructures that have a significant impact on the final prediction result becomes more difficult. Existing works on GNN explainability mainly focus on factual methods or counterfactual methods:

2.3.1 Factual Explanation

These methods aim to directly elucidate the influence of input features or substructures on the predictions. They can be further divided into Post-hoc and Self-interpretable:

Post-hoc. These methods are applied after the model training and explain already trained models without altering their structure. This technology is mainly divided into the following categories:

Decomposition-based methods explain a model's predictions by considering the output as a score and distributing it backward through the network layers to the input. The scores of different parts of the input are seen as their importance to the final prediction. These techniques require access to the model's internal parameters to calculate these scores. Examples include CAM [112], which uses the final node embeddings and one fully connected layer to learn the important score of a node in a specific classification task, but it can only be used in GNN models with a Global Average Pooling layer. Excitation-BP [112] distributes the output probability to neurons according to the ratios of their weights in excitations. Contrary to other methods, DEGREE [32] identifies influential subgraphs by decomposing message-passing mechanisms to find the contribution score of target nodes. GNN-LRP [117] uses a polynomial approximation to attribute importance to collections of edges within the graph.

Gradient-based methods focus on using gradients to represent the rate of change of predictions relative to inputs, which assess the prediction sensitivity to the changes in input features, thereby determining their importance. Among these methods, Sensitivity Analysis (SA) [5] uses the norm of the gradient of the model's predictions with respect to input features (either node or edge features) to quantify their importance. Another approach, Guided-BP [5], builds on SA but modifies it by clipping negative gradients during backpropagation to focus only on features that enhance the prediction, avoiding the potentially confusing effects of opposing gradient influences. Grad-CAM [112] builds on CAM but uses the gradients of the model's predictions with respect to the final node embeddings to quantify the importance of nodes in the network. This approach considers the final embedding of each node before the Global Average Pooling layer and computes node importance as the product of these embeddings and the classifier's weight vector for the target class. While the method assumes gradients as indicators of feature importance, it recognizes potential limitations like sensitivity, which may not always correspond to true feature importance, and the issue of saturation, where prediction changes are minimal despite variations in input, particularly affecting methods like SA and Guided-BP.

Surrogate methods are designed to explain complex models across a wide range of input values by fitting simpler, more explainable models in the locality of a specific prediction to capture the relationship between inputs and outputs. These methods usually generate a local dataset from the inputs near a prediction and then train a surrogate model based on this data to make explanations. For example, PGMExplainer [134] uses random perturbations to create a tabular dataset and Bayesian network that explains predictions, while GraphLime [53] employs a Hilbert-Schmidt Independence Criterion Lasso model for explainable N-hop neighbourhood node feature selection. Methods like RelEx [170] and DistilnExplain (DnX) [107] use GNN models as surrogates. The former requires an additional Breadth-first-search (BFS) sampling strategy to create the local dataset, which then uses a GCN model to fit it and a perturbation-based strategy to find a mask that acts as an explanation. The latter uses a more straightforward way to explain through a Simplified Graph convolution (SGC) [146] model distilled via knowledge distillation, making the explanation process more transparent.

Perturbation-based methods identify important subgraphs and node features by splitting input graphs into subgraphs and assessing their impact on model predictions. These methods use a subgraph extraction module to find influential subgraphs and a scoring function to evaluate the predictions made on these subgraphs against actual predictions. Techniques like GNNExplainer [158] learn and highlight the influence of subgraphs by masking. PGExplainer [85] and GraphMask [116] improve this process by using neural networks to parameterize edge distributions or masks, optimizing for mutual information between subgraphs and predictions. Other methods like Zorro [36], take a greedy strategy to select important nodes and features based on fidelity scores and approaches like SubgraphX [166] and GStarX [169] incorporate game-theoretic measures such as HN [42] values to evaluate importance, using Monte Carlo sampling for practical computation on large graphs.

Generation-based methods use generative models to create instance-level or model-level explanations for graph neural networks, ensuring graph validity through domain-specific rules or optimization constraints. For instance, XGNN [165] generates subgraphs that are influential for a certain class using reinforcement learning, while RG-Explainer [118] employs a similar RL approach for subgraph generation, optimizing for policy using cross-entropy loss as a reward. In contrast, GNNinterpreter [140] and GFLOW [75] use numerical optimization and flow matching conditions, respectively, to produce explanations without the need for handcrafted domain rules. GEM [78] applies Granger causality principles to deduce the causal importance of subgraphs, training a generative auto-encoder to provide explainable subgraphs.

Self-interpretable. These methods incorporate explainability directly within the GNN model, using informative subgraphs or node features that inherently provide explanations alongside predictions. Such methods derive explainability by incorporating interpretability constraints, which use either information constraints or structural constraints to derive an informative subgraph used for both the prediction and the explanation. Based on the design of the explainability, the self-interpretable methods can be classified into two types based on the imposed constraints.

In the realm of explainable AI, methods that generate explanations via subgraphs face the challenge of subgraphs being irregular and may have different sizes. Thus, merely constraining the size of the explanation may not be appropriate for the underlying prediction task. Information bottleneck (IB) [130] methods address this by applying information constraints rather than size constraints, optimizing the mutual information between the labels and the subgraphs while keeping the information within the subgraph under a certain threshold. GSAT [93] calculates variational upper bounds using stochastic attention, while others like GIB [161] use Donsker-Vardhan KL [25] representation for estimating mutual information in latent space, optimizing through bi-level training processes. VGIB [160] improves upon this by introducing noise to compress information, with a learned probability determining how much information from the original graph is retained, aiming to preferentially preserve information-rich substructures over those less relevant to the label.

Structural constraints are also fundamental self-interpretable methods for GNNs, dictating the extraction of informative subgraphs for prediction and explanation. DIR [149], an early method, discerns invariant causal rationales by segregating inputs into causal and non-causal elements, seeking a prediction that doesn't rely on the non-causal parts. ProtoGNN [171] employs prototype learning, matching new instances with learned prototypes for predictions and explanations. SEGNN [22] identifies similar structural and feature-based nearest neighbours for unlabelled nodes, aiding in prediction and deriving explanatory subgraphs. Meanwhile, KER-GNN [31] embeds graph kernels into GNNs to refine expressiveness and pinpoint significant substructures, using the similarity between nodes' subgraphs and trainable filters for this purpose. These methods share a common goal of enhancing model transparency by highlighting critical subgraph structures that rationalize the model's decisions.

2.3.2 Counterfactual Explanation

These generate alternative scenarios that could lead to different outcomes. By identifying minimal changes needed to alter predictions, these methods highlight critical data features that influence the GNN's decisions. Recently, there have been several attempts to generate explanations for graph neural networks (GNNs) via counterfactual reasoning, which can be classified mainly into three categories based on the type of methods:

Perturbation-based methods. Counterfactual generation for both graph and node classification is achieved by edge alterations to shift model predictions, often involving perturbations of the adjacency or computational graph. CF-GNNExplainer [84] introduces a binary mask matrix to perturb computational graphs, minimizing a loss that balances the accuracy of the counterfactual with its similarity to the original graph. CF2 [128] builds on this by optimizing for factual explanations and seeking subgraphs that are informative both factually and counterfactually. GREASE [17] adapts this concept for recommendation systems, where GNNs score rather than classify nodes, modifying perturbations to consider local graph neighbourhoods.

Neural framework-based methods. This section discusses neural architecture-based methods for generating counterfactual graphs where the adjacency matrix of the

input graph is minimally perturbed to generate counterfactuals. RCExplainer [4] uses graph embeddings to identify a resilient subset of edges whose removal alters predictions, aiming to reduce the fragility and potential overfitting of interpretations by clustering input graphs and targeting counterfactuals within these clusters rather than individual perturbations. In contrast, CLEAR [87] employs a graph variational autoencoder to generate entire graphs, addressing generalization and causality, and is capable of introducing new nodes, unlike RCExplainer, which is limited to existing graph structures. Both methods use generative models for their distinct objectives; RCExplainer focuses on the robustness of counterfactuals, while CLEAR aims to elucidate underlying causal relationships in the data.

Search-based methods. These methods employ search techniques over counterfactual space effectively, particularly when generative or perturbation methods fall short, such as in identifying active molecules from inactive ones in chemical reactions. Considering the potential enormity of this space, developing efficient search algorithms can be a challenge to identify valid and useful counterfactuals without being overwhelmed by the possibilities, ensuring practical applicability across various tasks and scenarios. For example, the MMACE [142] method utilizes the STONED [99] approach for graph classification tasks like assessing molecule solubility or blood-brain barrier permeation, employing BFS-style algorithms to pinpoint optimal counterfactuals. Alternatively, MEG [101] leverages reinforcement learning to efficiently traverse the search space in tasks predicting molecular toxicity or solubility. GCFExplainer [55] uses a different strategy using Vertex Reinforced Random Walks [106] to generate global explanations by finding a set of counterfactuals that represent broader subsets of inputs rather than individual cases. MMACE and MEG focus on predicting molecular properties with GNNs, employing different methods for exploring counterfactual spaces: MMACE uses a graph search algorithm, while MEG uses reinforcement learning for more efficient search. In contrast, GCFExplainer aims for global explanations and utilizes random walks to navigate the counterfactual landscape, highlighting a diverse approach to generating and accessing counterfactuals compared to MMACE and MEG.

2.3.3 Explainability in Query Optimization

Explainability in query optimization differs significantly from other domains where graph neural network (GNN) models are used. In query optimization, the goal is to represent and understand the underlying structure and cost behaviour of query plans, which are typically modelled as trees rather than general graphs. Although recent research has explored the use of learning-based models to enhance query plan representation and cost estimation, there is still limited work addressing the explainability of these models.

A unique challenge in this context stems from the structural characteristics of query plans. Query plans are inherently tree-based, and altering any part of the plan's structure can result in the loss of essential information from the subtrees beneath the modified node. Explanations based on subgraphs that lack important information can easily be inaccurate This sensitivity to structural changes makes standard GNN-based explainability techniques unsuitable for query plans.

Furthermore, query plan trees encode both the semantics and syntax of SQL queries, which makes it crucial to preserve their structural and semantic integrity during explainability. Current subgraph extraction techniques used in GNN explainability are not designed with these constraints in mind. As a result, applying these methods directly to query plan trees risks distorting the meaning of the query and compromising its explainability.

Despite these challenges, incorporating explainability into learning-based query optimization offers significant benefits. It enhances the reliability of query plan cost estimates, allowing database administrators and system users to identify which operations most influence cost predictions. This understanding can guide optimizations, leading to improved query performance. Given these advantages, developing tailored explainability methods for query optimization is a promising direction for future research, with the potential to bridge the gap between model interpretability and practical performance improvements in database systems.

2.4 Learning-based Query Optimization

Query optimization is a key process in database management systems that aims to generate and select query plans to determine the most efficient way to execute a given query. With the development of machine learning technology, researchers have gradually begun using machine learning-based models to replace traditional methods in query optimization to achieve performance improvements. In this section, we divide query optimization into five aspects and respectfully summarize and discuss the existing state-of-the-art learning-based studies in each aspect. The studies involved in Section 2.4.2 and Section 2.4.3 specifically refer to studies that only focus on the cardinality or cost estimation of the query plan but do not involve other components in optimizer such as plan selection.

2.4.1 Query Rewrite

Query rewriting is a technique used in databases to optimize user queries, converting them into a more efficient form for execution without changing their semantics. This process involves a deep semantic understanding of the queries, enabling rephrasing, reordering, simplifying the queries and optimizing operations like joins. The goal of query rewriting is to enhance database performance, improve resource utilization and obtain shorter execution times. The traditional method relies on heuristic rules defined by humans. On the one hand, the search space of all possible rewriting orders grows exponentially with the number of query operators and rules, and it is impossible to intelligently adapt search rules. On the other hand, predefined rewrite methods based on syntax-driven rules can easily fall into local optimal solutions.

To address these limitations, existing research has been exploring more advanced ways of query rewriting, hoping to replace traditional query rewriters with machine learning models, such as Sia [178], which optimize queries using learned predicates. The approach focuses on leveraging satisfiability modulo theories to generate counter-examples and use

them to iteratively learn a valid and optimal predicate [178], potentially improving performance by simplifying the computational requirements of database queries. [180, 181] employ combining Monte Carlo tree search [7] and learning-based query cost estimators to find the most efficient rewritten query from possible combinations. These studies show the shift from traditional heuristic-based methods to more dynamic, learned approaches that adapt to the characteristics of the data and queries, potentially leading to more efficient query processing in complex database environments.

2.4.2 Cardinality Estimation

Traditional cardinality estimation in query optimization usually relies on methods like histograms or predefined rules, which makes it difficult for these methods to handle complex queries or databases with irregular distributions or correlations between columns. Thus, modern methods are beginning to use machine learning techniques to predict the number of rows that match query predicates, resulting in more accurate cost estimates. Compared with traditional methods, these learning-based methods can use historical queries and their running workloads to train models so that they can adaptively learn the changes in data distribution and query patterns and provide more accurate and robust predictions. Existing cardinality method estimation can be mainly divided into the following two categories:

Data-driven Cardinality Estimators. Learned data-driven methods aim to describe the underlying data with machine learning models. Bayesian network (BN) based methods [38, 131] uses a directed acyclic graph to capture the dependence among attributes, assuming that each attribute is conditionally independent of others given the distributions of its parent attributes. BayesCard [150] uses probabilistic programming to improve BN's inference and model construction speed. In recent developments, machine learning approaches have been integrated into data-driven methodologies. Naru [157] and NeuroCard [156] employ deep autoregressive models to break down the joint distribution of attributes into a sequence of conditional distributions. DeepDB [50] is built upon Sum-Product Network (SPN) [110] which approximates the joint distribution using several local and simple PDFs. FLAT [183] improves SPN by adopting a factorize-split-sum-product network (FSPN) [152] to adaptively decompose the joint distribution according to the attribute dependence level. [136, 137] utilize the normalizing flow-based model to learn a continuous joint distribution for relational data, which transforms a complex distribution over continuous random variables into a simple distribution and uses the probability density to estimate the cardinality. LPLM [1] uses a new language and a novel probability distribution function to capture the semantics of general LIKE-patterns for improving cardinality estimation.

Query-driven Cardinality Estimators. Query-driven cardinality estimators are designed to capture the relationships between queries and their actual cardinalities. LW-XGB and LW-NN [30] see cardinality estimation as a regression problem, using gradient-boosted trees and neural networks for regression, respectively. UAE-Q [151] employs deep autoregression models and leverages differentiable progressive sampling through the Gumbel-Softmax trick to unearth latent details from queries. KDE-based join estimators [61] merge

kernel density estimation (KDE) with a query-driven tuning approach to approximate multivariate probability distributions of a relation and the cardinalities of joins. Fauce [82] and NNGP [172] presume that a query's cardinality follows a Gaussian distribution and use deep ensemble [67] and neural network Gaussian process [68] to obtain the relatively robust estimation based on the distribution's mean and variance. Another stduy [98] improves the robustness of the cardinality estimation model in different workloads through the ideas of random masking and Join Bitmaps. CEDA [141] improves estimation accuracy by automatically generating training workloads based on database data distribution and integrating histogram information into an attention-based cardinality estimator.

2.4.3 Cost Estimation

The application of machine learning technology to cost estimation in query optimization enables the prediction of the execution cost of database queries with greater accuracy and flexibility than is possible with traditional cost models. Unlike the training target of cardinality estimation, the learning-based cost estimation methods focus on the actual cost of query plan execution, including factors such as CPU time, I/O operations, or memory usage. In recent methodologies, a frequently used label for cost estimation is query plan latency, defined as the DBMS's runtime to execute a specific query plan, including all stages of query processing from start to finish. This is a critical metric in database performance, as it directly impacts the response time experienced by users. The learning-based cost optimization method targets the actual cost of the query plan and trains the model through historical query workload data. This enables the query optimizer to select the most efficient execution plan based on the predicted costs, thereby improving query performance and resource utilization.

Recently, the database community has been attempting to use deep learning models in cost estimation. The primary focus of existing deep learning models for cost estimation is analyzing query plans, especially their latency. [125] introduces a tree-structured model that learns the relationship between the query plan and its execution latency. QPPNet [91] also uses a tree-structured model but can predict the cost from the bottom up of the query plan. QueryFormer [174] employed Transformer [132] to encode the query plan representation for latency predicting. [49] proposed a Zero-Shot [153] cost model, which allows the pre-trained cost estimation model to be directly extended to unseen databases without fine-tuning. This approach significantly enhances the generalization ability of the cost model. Another study [83] developed an autoencoder based on LSTM and Self-Attention, which can unsupervisedly convert query plans into representations and assign pseudo labels to other samples based on the distance between query representations through a small number of labelled samples. This allows the model to achieve relatively good performance with only a small number of labelled samples in the training set and saves a lot of time required to generate samples. BASE [16] proposes a two-stage Reinforcement Learning [144] based framework to bridge the gap between cost and latency, allowing the model to achieve good performance in both cost and latency at the same time.

2.4.4 Join Order Selection

Join Order Selection (JOS) aims to find the most effective join execution order in SQL queries to minimize query execution time and resource consumption. Poor join order, especially when executing complex queries involving multiple tables, may cause the number of rows to be joined to grow exponentially with the number of relationships involved, resulting in longer execution times. Recent learning-based join optimization hopes to use machine learning technology, particularly reinforcement learning, to help a given query find the most effective join, providing a more dynamic and adaptable solution than traditional heuristic methods.

Existing learning-based JOS methods mainly focus on the field of using Deep Reinforcement Learning (DRL). ReJOIN [90] and DQ [66] are the first policy-based and value-based methods for JOS, respectively, both utilizing simple encodings for join tree representation despite their differing training data formats and learning approaches. Compared with ReJOIN and DQ, RTOS [163] further improves the JOS performance by integrating Tree-LSTM and a learning-based feature encoder to capture structural details of join trees and employing multi-task learning to jointly optimize cost and latency during different training phases.AlphaJoin [167] utilizes MCTS to replace the random search strategies in ReJOIN, DQ, and RTOS, which simulates various join orders within a single tree structure to identify the most promising ones, overcoming the limitations of random search that often misses optimal join orders. JOGGER [11] addresses the inefficiency issues of DRL methods by reducing the number of model parameters through graph-based models, including a Graph Convolutional Network and a tailored-tree-based attention module. This strategy enables JOGGER to achieve state-of-the-art performance with less optimization time and computational resources. SOAR [179] uses a Graph Neural Network and learns the inconsistent influence of the join tree on long-term reward with graph attention mechanisms in the reinforcement learning process so that it can match or outperform the optimizer in PostgreSQL.

2.4.5 Query Plan Selection

The studies involved in this section refer specifically to learning-based optimizer models, including query plan selection and models that aim to find the optimal query plan. Query plan selection in query optimization refers to the process of selecting the most effective one among multiple candidate execution plans for a given SQL query, which will significantly affect the execution speed and resource usage of the query. Traditional query plan selection relies heavily on traditional cost models, but due to the shortcomings of these models we discussed in Section 2.1, they are likely to bring inaccurate cost estimates when facing complex queries or databases.

Therefore, the learning-based query optimizer introduces machine learning technology into this process and implements the steps required for query optimization, including query plan enumeration, cost estimation, join order selection, etc., through learning-based methods. It conducts this process through specially designed query plan selection strategies

to find the most ideal candidate plan in the corresponding application scenario. Existing studies show that introducing machine learning into query optimization can improve the efficiency and accuracy of plan selection results.

As machine learning continues to evolve and apply to various aspects of query optimization, recent studies have shifted towards developing complete learning-based end-to-end query optimizers, among which Neo [89] has made a very significant contribution in this process. Neo proposed an end-to-end learning model for the first time, covering the query optimization steps from plan searching to selection. This model introduced a feature encoder designed to capture relationships at both the query and plan levels. It utilized a Word2vec-based [94] query predicate embedding, known as Row Vector, enabling the model to handle queries with previously unseen predicates, thereby enhancing its generalization capabilities. Neo was the first to develop a tree model based on TCNN [95] for learning query plan representations, which is still widely used today. Additionally, it built a value network from this tree model to estimate costs and select plans. The model also constructed a feedback loop so that it can be fine-tuned based on the actual execution latencies of user queries. Neo's integration of machine learning models in query optimization tasks allows it to learn and adapt from data automatically with advanced performance. Thus, the emergence of Neo has extensively promoted the development of learning-based query optimizers.

The research team of Neo later proposed Bao [88], which improved Neo by leveraging the inherent intelligence of existing query optimizers through query optimization hints and integrating TCNN [89] with Thompson sampling [129], a well-established reinforcement learning algorithm, to enhance its capabilities. This allows Bao to automatically learn from its errors and adapt to changes in query workloads, data, and schema. Compared with Neo, Bao can achieve better and more robust performance in shorter training time performance.

Both Neo and Bao initially rely on existing expert optimizers to generate workloads for model training. In contrast, Balsa [155], which is also based on deep reinforcement learning, begins its learning process with only basic knowledge acquired from a simple, environment-agnostic simulator. It then employs safe learning during actual query execution to further enhance its performance. Balsa represents the first demonstration that learning to optimize queries without input from an expert optimizer is both feasible and efficient.

In addition, different from the commonly used method of selecting the optimal query execution plan based on estimated cost, Lero [182] and Leon [15] innovatively used a ranking-based optimizer, employing a pairwise approach to train a classifier to compare any two plans and tell which one is better. Such a binary classification task is much easier than the regression task to predict the cost or latency [182], regarding model efficiency and accuracy. In other words, these methods focus on the order of input query plan pair's costs rather than precise cost values, reducing the effects of inaccurate cost estimations on plan selection and enhancing overall performance. The limitation of this method is that in comparing candidate plans of a query, pairs of candidate plans need to be used as input, which results in multiple iterations of comparisons being required to find the best plan. The sequential comparison process relies on the outcomes of previous comparisons

to proceed, which prevents parallel cost estimation of candidate plans and may lead to increased inference overhead.

Recent works [162, 27, 13, 15, 58] have begun to evaluate the uncertainty of the neural network models in the learning-based optimizer and workload data. The model or data's uncertainty or fluctuation caused by poor operators is considered through different strategies in the phase of plan selection. These methods make the decision-making process of the optimal plan not only based on the estimated cost but also consider these uncertain factors, which significantly improves the robustness of the model and reliability in selecting the optimal query execution plan in the real world.

2.5 Robustness in Query Optimization

Query optimization robustness has been explored through the following categories: runtime re-optimization, runtime parameter value discovery, robustness quantification, and the application of machine learning.

Re-optimization techniques. These methods utilize interval-based, rather than point estimates, to select plans that remain robust across specific parameter ranges [92, 2, 96]. These plans are used at runtime while actual parameters or more precise intervals are identified through execution or sampling. When discovered parameter values exceed the robustness range of the current plan, the optimizer recommends a new plan that is expected to be robust within the discovered range. Then, the execution engine prematurely stops the ongoing plan and switches to the new one. Such strategies require substantial modifications to extend both the execution plans and the execution engine to match the re-optimization process.

Discovery-based techniques. Discovery-based techniques avoid relying on parameter estimations by exploring the entire parameter space during compilation. They use strategies by iteratively executing plans within time budgets, discarding results if a plan exceeds its time allocation [29, 60, 59]. While these methods are theoretically effective, their compilation overhead is high, which makes them mainly suitable for OLAP-type environments. They also assume a perfect correlation between the optimizer's cost function and execution time and have to partially execute suboptimal plans to get actual parameter values.

Robustness quantification techniques. These techniques integrate considerations of robustness with optimality in plan selection. For instance, one method utilizes "Least Expected Cost" rather than "Least Specific Cost," employing a probability density function (PDF) of plan costs to enable selections based on expected costs across the parameter space rather than specific settings [20]. This method also considers cost variance as a factor in the cost function, allowing for a risk-adjusted approach to plan selection. However, analytically deriving cost distributions is complex due to the numerous parameters prone to error involved in such calculations. Another method assesses robustness using metrics like the derivative and area under the curve of the parametric cost function [145]. While this offers some objectivity, it fails to account for the probabilistic nature of estimates and

assumes uniform parameter distributions, which are unrealistic [48]. Additionally, it relies on classical cost models that may not be valid at runtime [19], potentially resulting in poor performance despite accurate parameter estimations. This method also incorrectly assumes that the cost function linearly correlates with cardinalities, which is often not the case in real-world scenarios.

Learning-based Techniques. Most learning-based query optimizer cost models focus only on achieving accurate cost estimates through supervised or reinforcement learning, usually using query plan cardinality or execution time as labels. While these methods generally show relatively good performance or generalization ability across various benchmarks, they are not explicitly designed to improve plan robustness and lack quantifying and leveraging the uncertainty inherent in the model's predictions and training data. However, some works have begun to address this problem, such as [162], which introduced a hybrid method that integrates learning-based cost models with traditional optimizers to select partial left-deep join orders based on predictive models. The approach also uses uncertainty estimates to discard plans with high uncertainty, defaulting to conventional optimization techniques when all plans exhibit high uncertainty. This study does not focus on maximizing robustness, nor does it showcase the robustness of its methodology. Additionally, it overlooks the possibility that plans with high uncertainty might still be more cost-effective than more stable plans and that pruning such plans might not always yield benefits. Overall, the improvements it offers over Bao are minimal.

Some recent research proposes solutions specifically for quantifying uncertainty. For example, some studies [82, 172, 173, 26, 58] introduce the Spectral-normalized Neural Gaussian Processes (SNGPs) [81] in cost model training so that these neural network models can estimate the variances of the target given the input queries while learning how to make accurate cost predictions. The variance obtained in this process is considered data uncertainty. To assess the uncertainty of estimator models, some studies [82, 173, 15, 58] employ Bayesian Neural Networks (BNN) [122] to determine the posterior distribution of model parameters and perform Monte Carlo Dropout [37] to generate multiple predictions and calculate expectations from the mean outputs for model uncertainty.

Based on the obtained data and model uncertainty, these methods can consider the magnitudes of the uncertainty when selecting the optimal plan to achieve significantly more robust performance. However, there is still a limitation in this process. The plan selection of the model based on the estimated cost and uncertainty remains independent of the training phase of the model. Current strategies often rely on predefined rules like using uncertainty thresholds to accept robust enough outputs or integrating the cost and uncertainty of candidate plans for selection. These optimization strategies that are also independent of the learning-based model make it unadoptable and challenging for the model to automatically learn from the results of plan selection and improve its uncertainty and cost estimation performance.

Chapter 3

Problem Statement

In this chapter, we outline the existing challenges and issues within the current field of research, providing precise definitions where necessary. Section 3.1 delves into the complexities surrounding query plan representation. Section 3.2 and Section 3.3 explore the interpretability mechanisms within query plans and the robustness of cost models, respectively. Lastly, Section 3.4 articulates the prerequisites for an ideal cost model, setting the stage for detailed discussions on these topics.

3.1 Query Plan Representation

In a query optimizer based on machine learning, query plan representation is a module that takes the physical query plan as input and uses a feature encoder and a tree model to generate vectors that subsequent machine learning models use for learning a downstream task. The generated query plan graph vector condenses important information about the physical query plan. At this stage, the information that subsequent models can learn is entirely derived from the query plan representations. In other words, the quality of these representations has a significant impact on the accuracy of cost predictions and can even determine the performance ceiling of the entire cost model. Therefore, enhancing the tree model's ability to accurately represent both node-level and structural information is crucial to improving the overall performance of the query optimizer. In this study, we focus on the representation capability of different query plan tree models in the same application scenario using the same structure of feature encoder and cost estimator so that the representation quality of a query plan can entirely rely on the tree model, allowing us to compare the performance of these tree models directly.

Due to the tree structure characteristics of the query plan, it is difficult for tree models to use traditional machine learning methods to learn and aggregate node information directly, for they usually cannot accept inputs in the form of trees. Consequently, the database community has started developing machine-learning models designed explicitly for tree structures [126, 95]. While the advanced models have marked performance improvements, they still have two main limitations:

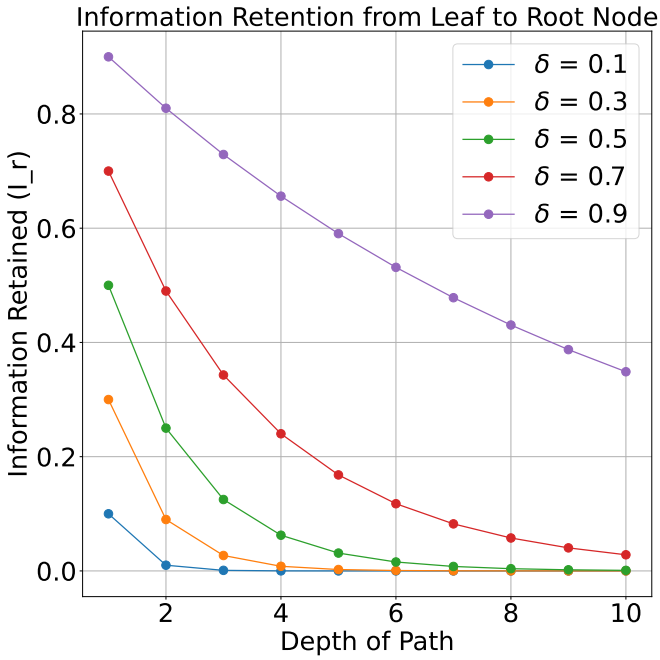


Figure 3.1: The impact of path depth on leaf node information dilution.

- **Information Dilution.** As information traverses from the leaf nodes—where specific relation-based operations are stored—towards the root, it tends to get diluted. This dilution process may negatively impact the accuracy of predictive cost estimations. The depth of the tree exacerbates this issue, leading to possible loss or dilution of critical information as it is passed upward or downward through the tree. The challenge of information dilution becomes more noticeable for complex queries that result in deeply structured trees. It is especially evident at the leaf nodes, which are the deepest in the tree but hold key relationship data critical for the model's estimates of basic cardinality.

  For instance, given a tree $T = (V, E)$ where V is the set of nodes and E is the set of edges, let $L \subset V$ represent the set of leaf nodes. Let $\text{path}(l, r)$ be the sequence of nodes from a leaf node l to the root node r, including l and r. Define a dilution factor $\delta_{u,v}$ for each edge $(u, v) \in E$, which represents the fraction of information retained when moving from node u to its parent node v. The total information retained from a leaf node l to the root r can be given by the product of dilution factors along the path:

$$I_r(l) = d_l \cdot \prod_{(u,v) \in \text{path}(l,r)} \delta_{u,v} \tag{3.1}$$

  where d_l is the initial information at the leaf node l. At this juncture, it is evident that the information retention originating from leaf nodes at the root node is influ-

enced by the dilution factor raised to the power of the path length. Assuming the initial information content at the leaf node is 1, and the dilution factor δ remains constant, the leaf node's retained information at the root node varies with the path depth, as depicted in Figure 3.1. As the path length increases, the remaining information transmitted from the leaf nodes decreases exponentially. This phenomenon necessitates careful consideration of information dilution when employing learning-based tree models for query plan learning. In scenarios involving complex and deep query plans, implementing mechanisms to retain or reinforce key information and avoid dilution becomes essential, ensuring that the final query plan representation comprehensively incorporates the critical data.

- **Preserving Structural Information.** The query plan tree encapsulates essential structural details, such as the combination of operations, their dependencies, and execution order, all of which significantly influence cost estimation. Preserving this complex structural information is critical when a tree model transforms the tree-structured query plan into a representation. However, due to inevitable information loss during the transformation, the performance of the cost model depends on the ability of the subsequent model to capture and reflect these remaining details through the representations and accurately assess the impact of different operation combinations and execution orders on cost prediction.

Therefore, the design of the tree model used for processing query plans should consider the above two limitations to further improve the model's representation ability according to the structural characteristics of the query plan tree. An effective and accurate query plan representation should maintain the structural information of the plan tree. It should minimize information dilution across node paths and incorporate detailed relational dynamics between nodes, thus enabling optimal decision-making by the query optimizer.

3.2 Explainability of Learning-based Cost Model

In traditional query optimizer cost models, which mainly utilize statistics-based methods such as histograms, the cost estimation for parent nodes relies on the estimated costs of their child nodes, progressing upward to calculate the total cost for the complete query plan. This approach ensures every part of the query plan's estimated cost is visible and transparent within traditional optimizers. Here, we refer to this capability to observe the impact of each part of the query plan on the final cost estimate as the "explain" functionality of the query optimizer cost model. However, introducing learning-based techniques into cost estimation brings a new challenge. Machine learning cost models often act as "black boxes," making it difficult to provide the "explain" functionality of cost estimations. They only focus on the accuracy of the final cost prediction results rather than the process of generating the results. This lack of transparency poses problems for researchers and database users who need to understand the internal model parameters and the decision-making processes within the model. Hence, identifying which specific predicates or operations in a query contribute to a given prediction becomes a complex task, hindering the ability to

analyze and optimize query performance effectively. Let G be a query plan graph, and let $C(G)$ denote the cost predicted by a learning-based cost model for G. Let $G_s \subset G$ be any subgraph of G. In this thesis, the explainability of the cost model is defined by its ability to provide a function F such that:

$$\text{Explain}(G_s, G) = F(G_s, G) \mid G_s \in G \tag{3.2}$$

where $\text{Explain}(G_s, G)$ quantifies how the presence and operations of G_s affect the overall cost prediction $C(G)$. The function F captures the contribution of G_s to the total cost, providing a quantitative explanation of its impact. The model's explainability is demonstrated by its ability to define this function F and accurately compute $\text{Explain}(G_s, G)$ for any subgraph $G_s \subset G$, enabling users to assess subgraph contributions and enhancing transparency in query optimization.

Existing research in query optimization tends to focus on model performance in terms of prediction accuracy but ignores research on the explainability of cost models. While mainstream tree models like LSTM, Tree-LSTM, and TCNN all belong to graph neural networks in a broad sense, the existing explainable mechanisms of graph neural networks cannot be directly applied to learning-based query plan representation. These explainability methods for GNN models typically employ strategies to extract subgraphs from samples and input them into the GNN model. They then interpret the characteristics of different subgraphs or their influence based on the predicted values produced by the model for these subgraphs. The main reason for its inapplicability in query plan cost estimation is that the subgraph extraction or search patterns of existing methods are not adapted to query plans, where obstacles arise from the property of query plans.

A query execution plan is typically structured as a tree, with its nodes representing a series of execution steps for the DBMS to retrieve the target data of the query. This structure implies that the prerequisite for the execution of a parent node is contingent upon the completion of all its child nodes and their returned results. This dependency leads to the fact that if a portion of the query plan is extracted as a subgraph without including all child nodes of the nodes within the subgraph, the subgraph lacks essential relationship and operation information. Incomplete subgraphs cannot function as valid subquery plans and lose key information, which makes the cost estimates generated by the cost model based on these incomplete subgraphs bound to be inaccurate and hard to explain the subgraphs' influence accurately. Explaining the impact of such an incomplete subgraph on the overall query plan's cost becomes irrelevant because it no longer represents a valid subset of operations from the original query. As illustrated by subgraph 3 in Figure 3.2, the extraction process did not capture all the child nodes associated with the involved node, leading to the omission of the scan operations for T1 and T2 from the original query plan, which is critical for the cost model to estimate the cost of the join operation. This omission resulted in an inaccurate cost estimate for all subsequent nodes in the subgraph and hindered the model's ability to precisely elucidate the impact of the subgraph on the prediction results based on its predicted value. Therefore, when extracting subgraphs from query plans, it is crucial to maintain the integrity of the tree model's tree structure.

Similarly, the query plan contains the semantics and syntax of its corresponding query and follows the query plan generation rules set according to DBMS. This necessitates that

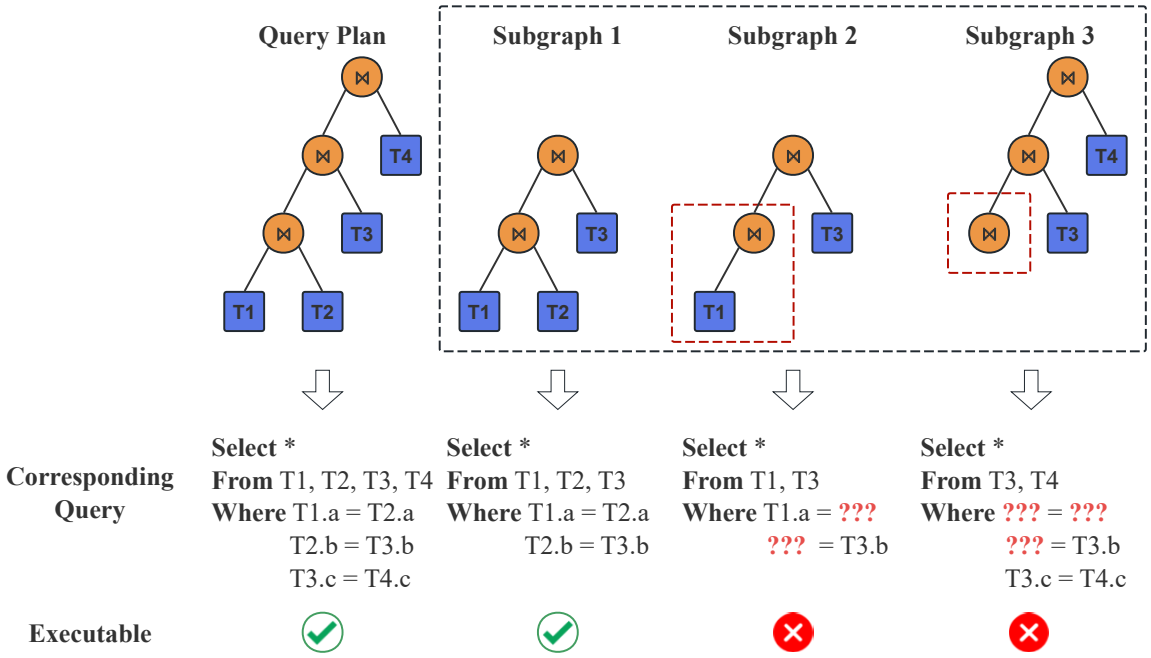


Figure 3.2: Examples of query plan subgraphs integrity.

any subgraph extracted from the query plan must retain complete semantics and be executable. The subgraph can be actually executed by the DBMS without error, which ensures that the cost model can treat the subgraph as a separate valid sample and make relatively accurate predictions. For instance, as shown by subgraph 1 and subgraph 2 in Figure 3.2, a join node in a query plan requires two child nodes to supply the necessary relations for the join node, whereas subgraph 1 maintains a complete structure and accurately reflects the corresponding query semantics, enabling it to be executed as a legitimate sub-plan of the original query. In contrast, Subgraph 2 is missing a child node, which distorts its corresponding query semantics and renders it inexecutable. This lack of functionality means that it incurs no practical cost, rendering any cost predictions for it unhelpful for explanation purposes. Missing nodes in the subgraph extracted from the query plan can lead to deviations or even errors in the corresponding query semantics. Therefore, during the explanation process of learning-based query plan representation, it is necessary to ensure that the extracted subquery plan itself follows its construction rules and maintains its corresponding original query semantics. In this thesis, structural integrity refers to preserving the complete tree structure of a query plan to ensure its executability by a DBMS, maintaining all relevant nodes and edges. An effective subgraph extraction strategy maintains this structural integrity, allowing extracted subgraphs to serve as valid, standalone samples for accurate cost predictions and effective explanations of subgraph influences.

To effectively implement an explainability mechanism in the learning model used for query plan representation and cost estimation, a method is required to ensure the extracted query plan subgraph has a complete structure and correct corresponding query semantics. Additionally, this mechanism must be able to explain the contribution of specific operations

within the query plan to the prediction results and can be integrated into the original cost model architecture without losing the model's performance.

3.3 Robustness in Learning-based Cost Model

Most query optimizers prioritize the accuracy of cost estimates in their design, often overlooking the inherent uncertainties. These cost models usually simplify assumptions about the data and operational environment, assuming that no perturbations occur during the cost estimation process. However, the reality is that uncertainties in query plan execution can be significantly influenced by factors like structural characteristics, the operations or predicates used, and underlying data characteristics. Similarly, the cost estimation model itself is likely to be influenced by inaccuracies in parameters and non-conforming environments. In this thesis, robustness in query optimization is the ability of a cost model to maintain accuracy and be insensitive to the inherent uncertainties in the operational environment, model parameters, and variations in structural and data characteristics.

Therefore, how to quantify these uncertainties and apply them in query optimization to achieve more robust performance presents a challenge. Current state-of-the-art robustness-based research can already achieve quantification of model uncertainty through methods like Bayesian neural networks or Monte Carlo techniques and use Spectral-normalized Neural Gaussian Processes for query plan data uncertainty quantification to enhance robustness. However, these methods have a significant limitation: they can indeed quantify uncertainty and employ various strategies during the plan selection phase to balance estimated costs and uncertainties of candidate plans and find a more robust and efficient plan. However, the plan selection part of these methods is not included in the training process of the cost model, which means that the model cannot adaptively optimize its accuracy of cost estimation, uncertainty quantification and plan selection strategies through the results of plan selection. Therefore, how to integrate the intelligent balance of cost estimation and uncertainty quantification and utilize the comparison results from the plan selection phase into the training process of the cost model remains a problem we need to solve.

3.4 Problem Definition

In summary, a more accurate, robust and explainable query optimization cost model necessitates a comprehensive framework that generates query plan representation accurately, explains the influence of each subgraph of the query plan on the cost predictions, and automatically learns to quantify uncertainty for improving robustness and applies it during plan selection, and integrates the plan selection results in model training. Specifically, the framework should fulfill the following requirements:

- Compare and analyze the performance of mainstream tree models on different tasks
- Provide techniques to achieve more accurate query plan representation

- Provide techniques to explain cost model prediction results
- Provide quantification functions for uncertainty
- Provide learning-based strategies to combine quantified uncertainty and cost estimation for plan selection
- Provide techniques to enable cost model learning and self-improving from the plan comparison results in the plan selection phase

Chapter 4

Framework Overview

In this context, in order to overcome the limitations of existing learning-based cost models for query optimization, we have proposed three targeted techniques designed to effectively improve current limitations and enhance the cost model's performance in three aspects: plan representation, explainability, and robustness. This chapter is structured as follows: Section 4.1 introduces our novel proposed tree model for query representation based on bidirectional GNN and GRU technologies. Section 4.2 describes our general explainability technique tailored for learning-based query optimization cost models. Lastly, Section 4.3 presents our design of a learning-to-rank, uncertainty-aware cost model.

4.1 BiGG: A Novel Technique for Query Plan Representation Based on Bidirectional Graph Neural Networks

The basic idea of a GNN network is that nodes can capture the characteristics and contextual information of neighbour nodes and their complex relationships via edges in the graph. Acknowledging the benefits of GNN, due to the tree-structured query plan's characteristics discussed above, existing GNN methods do not perform well in query plan representation task scenarios. Thus, we propose a new tree model based on bidirectional GNN and GRU, which effectively addresses the following two main issues:

1. Information Transmission between tree nodes (Section 4.1.1)
2. Information Aggregation from node-level to graph-level (Section 4.1.2)

4.1.1 Bidirectional Graph Neural Networks

In the existing tree models for query planning, two primary methods of information transmission are mainly employed. The first method involves transferring all nodes' information

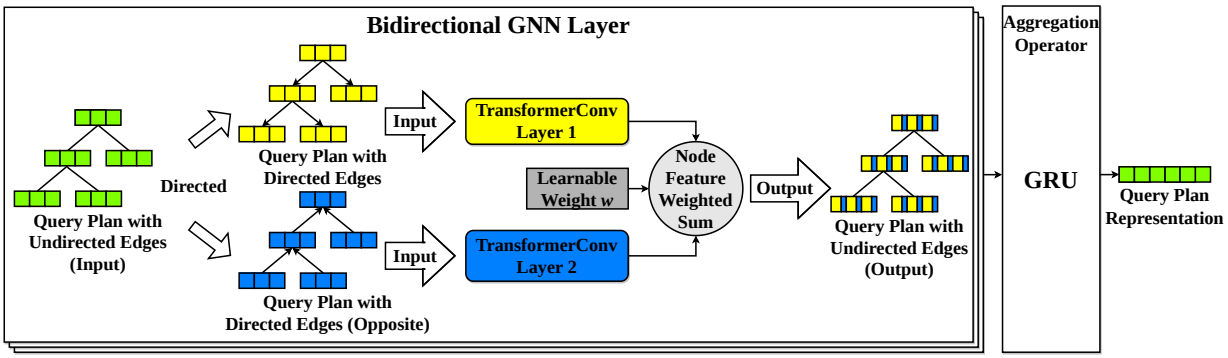


Figure 4.1: The architecture of tree model using bidirectional GNN with aggregation operator GRU.

to the root node. This allows the root to aggregate the entire graph's data, thereby enabling its features to represent the entire graph, as seen in models like Tree-LSTM [126]. The second method treats each node as both a source gathering information from its parent or child nodes, then employs a global pooling mechanism to form graph-level representations, such as Tree-CNN [95]. However, these methods are not directly applicable in GNN due to the unique characteristics of message passing in these networks.

In GNN, message passing between nodes depends on both the existence of edges in graphs and their directionality. In the tree structure, this directionality issue is mainly divided into two situations: single-directed and undirected edges. If single-directional edges are used, such as from the child node pointing to the parent node, information is always transmitted from the child node to the parent node. A node in GNN can only learn the information of its adjacent nodes in one layer, which means any node in the tree only learns information about its child nodes. This limited scope of learning becomes a problem when attempting to transfer information from the leaf nodes to the root in deeper trees. For example, a number of GNN layers equivalent to the tree's depth is required, which is often impractical due to computational constraints and the risk of overfitting. Additionally, the single-directed edge also makes it challenging to transfer the leaves' information to the root node, passing the entire graph without diluting or losing it, which makes using root node features as query plan graph-level representation infeasible. Furthermore, this directional bias prevents child nodes from ever learning about the features and structural information of their parent and higher-level nodes. Consequently, valuable structural information is inevitably lost during the global pooling for node feature aggregation.

Therefore, using undirected edges has the potential to significantly speed up the information transfer between nodes, but they may destroy the structural relationship between parent and child nodes and make the model unable to learn the hierarchical dependency between them. Therefore, in order to speed up information transfer and preserve the hierarchical relationship between nodes, inspired by DirGNNConv [114], we innovatively treat the query plan trees as two one-way graphs with opposite edge directions. In each layer of our tree model, as shown in Fig. 4.1, we divide the original query plan into two tree graphs with opposite edge directions as input to two independent TransformerConv [121] layers respectively and integrate the corresponding node features from the two output learned

graphs through a learnable parameter, which makes it possible to transmit information in both directions while still utilizing the direction information of the edges and retaining relevant structural information. Given a query plan tree graph G with N nodes, n as the nodes in the node feature set $Nodes$, E as the undirected edge index set, $E_{\text{child-to-parent}}$ and $E_{\text{parent-to-child}}$ as the edge index sets with opposite directions, p as a learnable parameter, the bidirectional GNN layer is defined as Algorithm 1:

Algorithm 1 Bidirectional GNN Layer Update

Input:

$G = (N, E)$: query plan tree graph with nodes N and edges E

Node features $\{\mathbf{h}_n^{(l)} \mid n \in N\}$ at layer l

Edge sets with opposite directions:

$E_{\text{child-to-parent}}$: edges from child to parent nodes

$E_{\text{parent-to-child}}$: edges from parent to child nodes

$p \in [0, 1]$: a learnable parameter

Output:

Updated node features $\{\mathbf{h}_n^{(l+1)} \mid n \in N\}$

Update node features in the child-to-parent direction:

$\{\mathbf{h}_n^{\uparrow} \mid n \in N\} \leftarrow \text{TransformerConv}_{\uparrow}\left(\{\mathbf{h}_n^{(l)}\}, E_{\text{child-to-parent}}\right)$

Update node features in the parent-to-child direction:

$\{\mathbf{h}_n^{\downarrow} \mid n \in N\} \leftarrow \text{TransformerConv}_{\downarrow}\left(\{\mathbf{h}_n^{(l)}\}, E_{\text{parent-to-child}}\right)$

Integrate the node features:

for each node $n \in N$ **do**

    $\mathbf{h}_n^{(l+1)} \leftarrow p \cdot \mathbf{h}_n^{\uparrow} + (1 - p) \cdot \mathbf{h}_n^{\downarrow}$

end for

Figure 4.2 shows the impact of varying edge directions on the GNN model that utilizes the TransformerConv [121] layer for message passing and global add pooling for aggregation. Subgraphs (a) through (d) display cost estimation results for the same query plan, processed by GNN models trained and tested on identical workloads but with different edge directionalities. These directionalities include undirected edges, single-directed edges (from root to leaf and vice versa), and our bi-directed weighted edges. Each node within the shown query plan trees captures several details: the node's operation type, the actual execution time ET_{actual} and the GNN model estimated execution time $ET_{estimated}$ of the subtree for which the node serves as the root, and the corresponding Q-error value, which measures the accuracy of estimated versus actual execution costs, defined as follows.

$$Q\text{-error} = \frac{\max(ET_{actual}, ET_{estimated})}{\min(ET_{actual}, ET_{estimated})} \tag{4.1}$$

This setup allows us to assess the capacity of the GNN models trained based on the complete query plan for estimating the partial query plans of different sizes within the plan and to explore how different edge directions influence model performance. Figure 4.2(a)

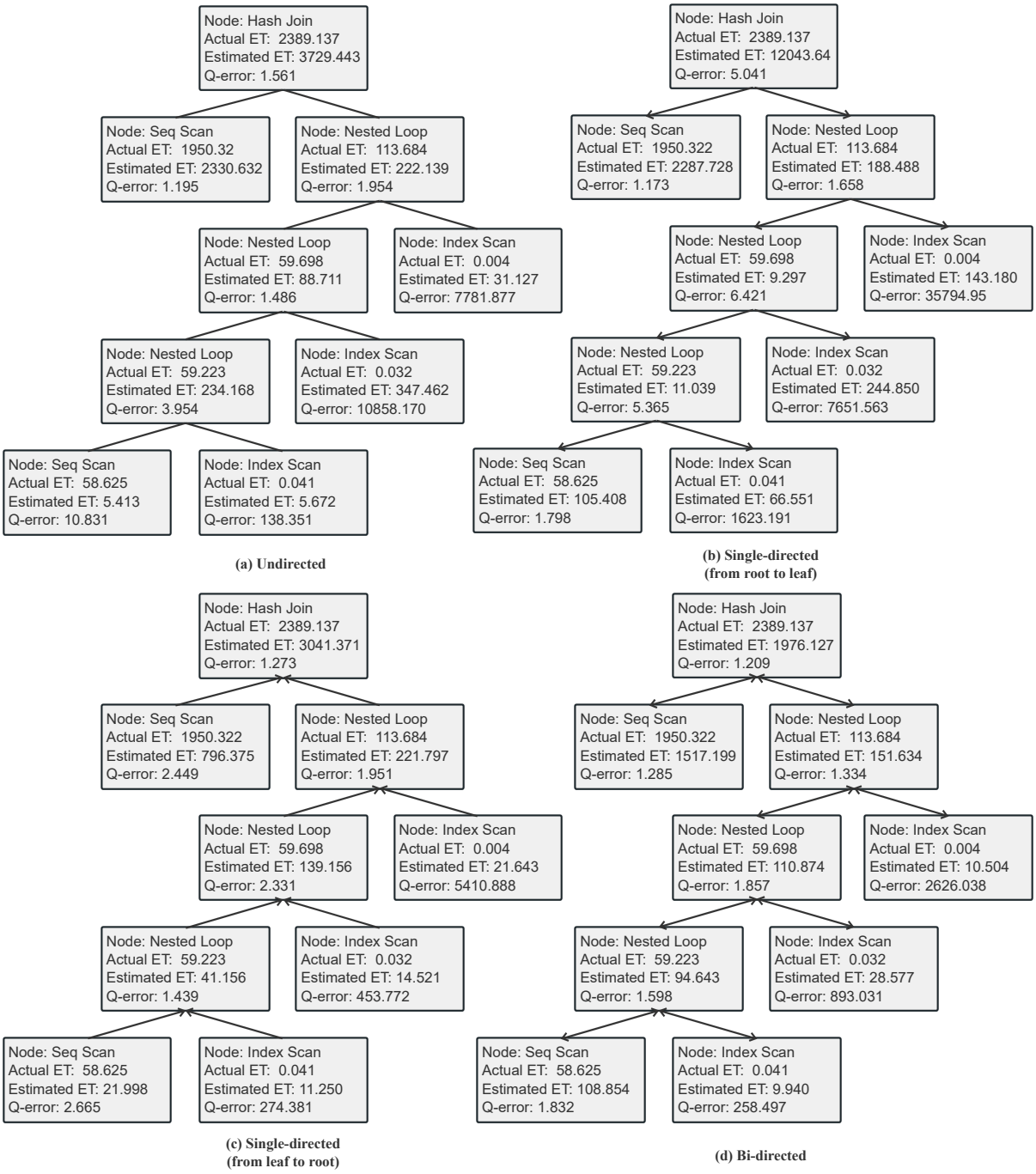


Figure 4.2: Examples illustrating the impact of different edge directionalities on GNN models in query plan cost estimation.

highlights that GNN models trained with undirected edges provide more accurate cost estimates for subtrees with intermediate nodes as roots, compared to estimates for the entire query plan or leaf nodes. This is because undirected edges allow nodes to simultaneously receive information from both children and parent nodes, enhancing message passing. Due to the characteristics of the tree structure, the nodes close to the root node or leaf node are close to the boundary and have a larger depth, resulting in the nodes being able to obtain relatively less information. In contrast, the intermediate nodes can generally collect information from both the top and bottom and fully exchange information, making their embeddings rich in information. This implies that during global aggregation across all nodes, the embeddings are predominantly influenced by the intermediate nodes, which contain a larger amount and are richer in information. Consequently, the information on the root and leaf nodes becomes relatively diluted. As a result, the GNN model utilizing undirected edges achieves a more accurate understanding of subtrees where intermediate nodes are the root, but it is less effective at accurately estimating the costs for the entire query plan or leaf nodes. Meanwhile, the use of bidirectional edges prevents the model from learning the dependency relationships between parent and child nodes, thereby limiting its overall accuracy.

Figure 4.2(b) demonstrates the impact of using a single directed edge (from the root to the leaf node) on the GNN model. This configuration results in less accurate cost estimates for subtrees rooted at the root or intermediate nodes compared to other models, while the accuracy for relatively time-consuming leaf nodes is better. At this time, when the query plan node information is transmitted in the GNN model, since the direction of the edge is from the root node to the leaf node, the information can only be transferred from the root node to the leaf node. While other nodes, especially those close to the root node, cannot learn the key operations or relationships contained in all their child nodes. Instead, their information is gradually diluted in the information transmission. Consequently, leaf nodes, which accumulate the most information, disproportionately influence the global aggregation embedding. The results show that using this kind of single-directed edges enhances the model's understanding of leaf nodes, but it does so at the expense of reduced accuracy for other parts of the query plan.

Figure 4.2(c) illustrates the effects of using a GNN model trained with directed edges from the leaf nodes to the root node, whose information flow in the query plan tree at this time is the most consistent with the execution order of the query plan that is actually implemented. This configuration achieves the most accurate cost estimates for the entire query plan but tends to be less precise for leaf nodes. With this edge direction, information flows upward from the leaves to the root. Consequently, during global aggregation, the upper nodes that collect more information become more dominant in the global embedding, leading to a relative dilution of the lower nodes' data. This enhances the model's ability to estimate the overall query plan but diminishes its accuracy for smaller or lower-level subtrees. Furthermore, while the shallow depth of the query plan tree in this example allows the root node to integrate information from the lower levels effectively, a deeper and more complex tree structure might prevent the root from learning critical data from the bottom nodes. This limitation could significantly impair the model's performance by diluting essential information during aggregation.

This demonstrates that varying edge directions significantly impact the GNN model's understanding of different parts of the query plan and its effectiveness in processing various sizes of the query plan. Specifically, employing undirected edges enhances the model's comprehension of intermediate nodes. Conversely, using directed edges from the root node to the leaf node and from the leaf node to the root node improves cost estimation accuracy for leaf nodes and root nodes, respectively.

Therefore, as illustrated in Figure 4.2(d), the GNN model we propose, which utilizes bidirectional weighted directed edges, effectively balances the characteristics of the three edge directions. This design allows information to converge towards both the root and leaf nodes simultaneously, preventing excessive dilution of node information at both ends. Additionally, its direction weighting ensures that the GNN model can learn the hierarchical relationship and dependency between different nodes. From the example, the model's estimation accuracy for leaf nodes is close to the model with single-directed edges from the root to the leaf, and it surpasses other models. For subtrees where an intermediate node serves as the root, its performance is comparable to or even exceeds that of the model using undirected edges. Moreover, its overall query plan estimation matches the accuracy of models using a directed edge from the leaf node to the root and outperforms other configurations. By integrating the strengths of the three edge directions, our bidirectional GNN model enhances its cognitive abilities for comprehensively understanding the entire query plan and various internal subplans, which makes our GNN model applicable to the cost estimation of the query plan.

Additionally, the TransformerConv we employ as the GNN message passing layer in this proposed architecture adopts the Transformer's multi-head attention [132] in graph learning. Specifically, given node features $H^{(k)} = \{h_1^{(k)}, h_2^{(k)}, ..., h_n^{(k)}\}$ as the input to k-layer, the graph multi-head (C-heads) attention for each edge from j to i is given by:

$$
\begin{aligned}
q_{c,i}^{(k)} &= W_{c,q}^{(k)} h_i^{(k)} + b_{c,q}^{(k)} \\
k_{c,j}^{(k)} &= W_{c,k}^{(k)} h_j^{(k)} + b_{c,k}^{(k)} \\
e_{c,ij} &= W_{c,e} e_{ij} + b_{c,e} \\
\alpha_{c,ij}^{(k)} &= \frac{\langle q_{c,i}^{(k)}, k_{c,j}^{(k)} + e_{c,ij} \rangle}{\sum_{u \in \mathcal{N}(i)} \langle q_{c,i}^{(k)}, k_{c,u}^{(k)} + e_{c,iu} \rangle}
\end{aligned}
\tag{4.2}
$$

Where $\langle q, k \rangle = \exp\left(\frac{q^T k}{\sqrt{d}}\right)$ is recognized as an exponential scale dot-product function, where d signifies the dimensionality of each attention head. For each c-th attention head, the source feature $h_i^{(k)}$ and the corresponding neighbor node feature $h_j^{(k)}$ are transformed into the query vector $q_{c,i}^{(k)} \in \mathbb{R}^d$ and the key vector $k_{c,j}^{(k)} \in \mathbb{R}^d$, respectively. These transformations are achieved using different trainable parameters $W_{c,q}^{(k)}$, $W_{c,k}^{(k)}$, $b_{c,q}^{(k)}$, and $b_{c,k}^{(k)}$. Edge features e_{ij} are also encoded and incorporated into the key vector, providing additional contextual information for each layer. Subsequently, the message aggregation from the neighbor node j to node h_i is given by:

$$
\begin{aligned}
v_{c,j}^{(k)} &= W_{c,v}^{(k)} h_j^{(k)} + b_{c,v}^{(k)} \\
\hat{h}_i^{(l+1)} &= \Big\|_{c=1}^{C} \Big[\sum_{j \in \mathcal{N}(i)} \alpha_{c,ij}^{(k)} (v_{c,j}^{(k)} + e_{c,ij}) \Big]
\end{aligned}
\tag{4.3}
$$

Where the $\|$ is the concatenation operation for C head attention. Therefore, each node can adaptively aggregate information from different neighbouring nodes using multi-head attention [132]. In our tree model, this functionality allows each node to independently assess and learn the attention of its child nodes and parent nodes separately, enabling it to carry out adaptive weighted aggregation and further improve the model's ability to capture the tree's local graph topology and global dependencies. This adaptation enables the model to weigh the importance of different node operations and relations relative to each other, effectively handling the representation learning of the two single-directed query plan tree graphs. Therefore, by the weighted bidirectional GNN, we can improve the Information Transmission issue and further enhance the performance of the model in representing the tree-structured query plan.

4.1.2 Aggregation Operator based on Gated Recurrent Units

Conventional graph aggregation methods often produce inferior results when processing tree structure graphs. These methods usually simply aggregate node features by global pooling, which ignores the structural information in the graph. In our model, we innovated by applying GRU [9] to aggregate the GNN-learned query plan nodes after the tree has been post-traversed.

The Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that is designed to manage long-term dependencies by controlling the flow of information through gates, making them well-suited for processing the post-traversed query plan where the execution of subsequent nodes depend on the previous nodes. It introduces gating units to control the flow of information, addressing the vanishing gradient problem present in traditional RNNs. Compared with LSTM, GRU simplifies the architecture by combining the forget and input gates into a single update gate and merging the cell state with the hidden state, resulting in fewer parameters and potentially faster training while retaining similar performance. The architecture of GRU is shown in Figure 4.3. Specifically, given the t-th node feature x_t in the post-traversed query plan node sequence, GRU consists of an update date z_t that controls how much of the past information needs to be passed along to the future and a reset gate r_t determines how much of the past information to forget. Then, the input hidden statement vector h_{t-1} from the previous unit, the output hidden statement vector h_t containing the information up to this node is given by:

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \tanh(W_h x_t + U_h (r_t \cdot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \cdot h_{t-1} + z_t \cdot \hat{h}_t
\end{aligned}
\tag{4.4}
$$

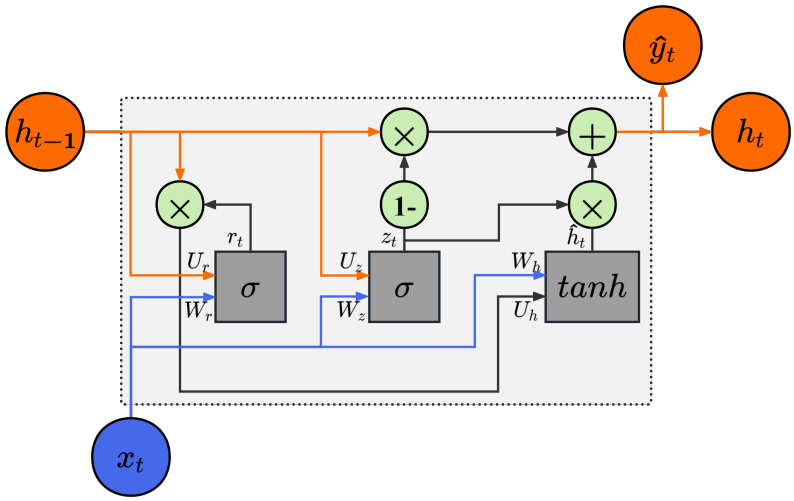


Figure 4.3: The architecture of Gated Recurrent Unit (GRU).

Where $\hat{h}_t$ is the candidate activation that represents the new content to be added to the hidden state, influenced by both the current input and the reset gate. σ denotes the sigmoid activation function, and tanh represents the hyperbolic tangent activation function. The matrices W_z, W_r, W_h are the weights associated with the input x_t, while U_z, U_r, U_h are the weights associated with the hidden state h_{t-1}. The vectors b_z, b_r, b_h are the biases for each gate.

The rationale for using GRU as the GNN aggregation operator is based on the observation that the order in which DBMS executes the query plan tree nodes and the order of nodes obtained by post-order traversal of the tree are similar [83]. This means after converting the post-traversed query plan tree into a sequence of nodes, and the sequence is used as input for the GRU model. The order in which nodes are input into the model closely resembles the sequence in which they are executed within the DBMS, enabling the GRU to intelligently determine when node features should be remembered or forgotten and how different operations impact the final prediction. This design learns the dependencies between nodes while more closely conforming to the actual execution sequence, thereby allowing our tree model to obtain the query plan's graph-level embedding while retaining the node features and structural information in the query plan tree to a certain extent.

4.1.3 Proposed Tree Model Architecture

As shown in Figure 4.1, our innovative tree model architecture, which combines weighted bidirectional GNNs with a GRU aggregation operator, effectively addresses the primary limitations of representing tree-structured graphs. The model not only retains the integrity of node and structural information in the query plan but also leverages GRU to adaptively learn and aggregate node information in a sequence that closely mirrors actual execution. This enhancement significantly boosts the model's capacity to handle complex queries, resulting in a more accurate and efficient representation of query plans.

4.2 An Explainability Technique for Learning-based Cost Model Based on Subtrees

In query optimization, a cost model that can explain its decision-making process, specifically one that can explain the influence of certain subplans or specific operations or predicates on the prediction outcomes, can significantly improve human transparency and trust in the model. This improved understanding allows for more targeted optimizations of queries based on the insights derived from the decision-making process. However, existing studies have largely overlooked the development of interpretable mechanisms for cost models that are based on query plan trees. Therefore, in this section, we propose an innovative explainability technique designed for query optimization cost models based on the characteristics of query plans, which can be seamlessly integrated into existing cost models.

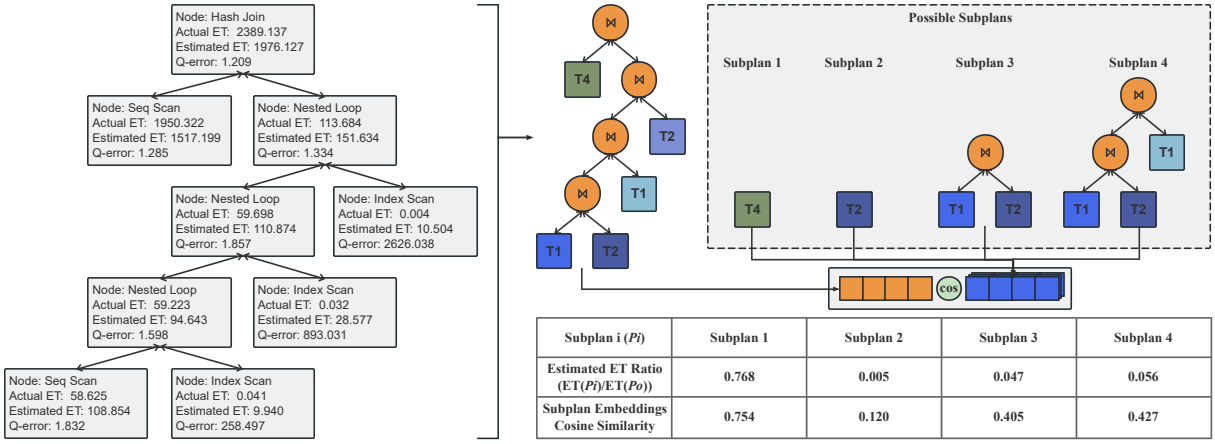
4.2.1 Subgraph Extraction Based on Query Plan Subtrees

Though query plans are also graphs, current explainability mechanisms designed for GNN models can not be directly applied to cost models for query plans. The main reason for this is, as we discussed in Chapter 3, the special structure and implicit semantics of query plans cause existing subgraph extraction patterns unsuitable for the query plan representation learning.

In order to avoid destroying the inherent structural integrity of the query plan, our observations indicate that if a node in the query plan and all its child nodes are included in the extracted subgraph, it ensures that the subgraph can be executed under a DBMS. The extracted subgraph is actually a subtree formed with any given node as the root extending down to the leaf nodes of the original query plan. Such a query plan subtree can be actually executed by the DBMS, which maintains the structural integrity and complete query semantics, enabling the tree model to effectively learn and generate representations. This method allows for segmenting the original query plan into subtrees of varying sizes, ranging from the smallest consisting of a single leaf node, and progressively including more levels up to the entire original query plan, as shown in the upper half of the Figure 4.5, where omits the situation where the leaf nodes are separate subtrees due to space constraints.

In practice, when a DBMS like PostgreSQL executes a node from the query plan, it records the cumulative runtime of the node and all its child nodes, termed the total actual time of the node, which supports us to obtain the actual runtime for specific subgraphs:

- If a subgraph that needs to be explained forms a complete subtree (even if it's just a single leaf node), the total actual time of its root node accurately represents the runtime needed to execute all operations within that subgraph;
- If the subgraph is an incomplete subtree or in an irregular form, we can use the total actual time of the larger subtree containing the subgraph to subtract the summed total actual time of the remaining one or more encompassing subtrees after removing the subgraph from the larger subtree, which allows us to obtain the precise runtime assessment of even irregularly structured subgraphs.



Subplan i (*Pi*)	Subplan 1	Subplan 2	Subplan 3	Subplan 4
Estimated ET Ratio (ET(*Pi*)/ET(*Po*))	0.768	0.005	0.047	0.056
Subplan Embeddings Cosine Similarity	0.754	0.120	0.405	0.427

Figure 4.4: Example of the relationship between the cosine similarity of subquery plan embeddings and the overall query plan embedding, and their influence on the cost prediction result.

4.2.2 Learning-based Explainability Technique

In the study of GNN explainability, literature [86] has shown that real-life graphs have underlying structures, which assumes that if an underlying subgraph significantly influences the final prediction result, there must be a discernible correlation or similarity between the embedding of the subgraph and the embedding of the complete graph so that the prediction model can make a similar decision. Let G be a graph with its corresponding graph-level embedding Emb_G derived from a GNN model. Let $G_s \subset G$ represent a subgraph of G with its own embedding Emb_{G_s}. The influence of G_s on the predictions made by the GNN for G is usually quantified by the correlation or similarity between Emb_{G_s} and Emb_G. To identify the subgraphs that most significantly influence the final prediction, the problem is framed as ranking all subgraphs in descending order based on their similarity with Emb_G, reflecting their impact on G's overall prediction:

$$\text{Rank}(\text{Emb}_{G_s}) = \text{sort}_{\text{descending}}\left(\text{similarity}(\text{Emb}_{G_s}, \text{Emb}_G)\right) \tag{4.5}$$

Where similarity is a function measuring the similarity between the two embeddings, such as cosine similarity or mutual information (MI) [65] and other relevant metrics. $\text{Rank}(\text{Emb}_{G_s})$ indicates the subgraphs' ranking based on their similarity, with higher values always signifying a greater influence on the model's predictions.

This assumption can also be applied to the explanation of query plan cost estimation. In Figure 4.4, we show the application of this assumption through an example that uses the same query plan tree as in Figure 4.2, with each node containing the model estimated execution time of the sub-plan with the node as the root node. We extracted several possible subplans from the query plan to explore and demonstrate the relationship between the cosine similarity of the subplan embeddings and the overall plan embedding and its influence on the final prediction result. Here, the ratio of the model's estimated execution time for the subplan $ET(P_i)$ to the estimated execution time for the original plan $ET(P_o)$

is regarded as the influence of the sub-plan on the final cost estimation result, and the similarity between the sub-plan and the complete plan embedding is quantified by cosine similarity. The cosine similarity between the query subplan and complete query plan embeddings Emb_{sp} and Emb_p can be defined as:

$$\text{cosine\_similarity}(Emb_{sp}, Emb_p) = \frac{Emb_{sp} \cdot Emb_p}{\|Emb_{sp}\|\|Emb_p\|} \tag{4.6}$$

Where $Emb_{sp} \cdot Emb_p$ is the dot product of the vectors, and $\|Emb_{sp}\|$ and $\|Emb_p\|$ are the norms (magnitudes) of vectors Emb_{sp} and Emb_p, respectively. Cosine similarity effectively measures the cosine of the angle between their vectors, providing a scale-independent measure that indicates how closely related the directions of their embeddings are, regardless of their magnitudes. Thus, this can express the correlation or similarity between two query plan embeddings to some extent.

From the right half of Figure 4.4, we can observe that subplan 1 has the greatest influence on the estimated cost of the entire query plan despite containing only one leaf node. This sub-plan accounts for a substantial portion of the model's estimated execution time, with a ratio of 0.754 to the total estimated execution time of the original plan. Correspondingly, its cosine similarity to the complete plan's embedding is 0.768, the highest among all subplans. In stark contrast, subplan 2 minimally impacts the overall cost estimate, contributing just 0.005 to the estimated execution time. Its cosine similarity value of 0.120 indicates a weak relationship with the complete plan's embedding, aligning with our assumption.

Subplans 3 and 4 are larger and exhibit an inclusion relationship, where subplan 4 includes an additional nested loop join and index scan node operation compared to subplan 3. Consequently, the estimated cost for subplan 4 should be higher than that for subplan 3, as reflected in their estimated execution time ratios of 0.047 and 0.058, respectively. The cosine similarities of 0.405 for subplan 3 and 0.427 for subplan 4 are consistent with these trends, suggesting that subplan 4's embedding is more closely aligned with the complete plan's embedding compared to subplan 3, yet it is less influential than subplan 1 and significantly more so than subplan 2. These findings indicate a positive correlation between the cosine similarity of a subplan's embedding and its share of the estimated execution time, further validating that the assumption can be applied to query plan cost estimation. To capture the relationship or similarity between subplans and the complete query plan more accurately and adaptively, we propose a learning-based explainability technique for learning-based cost models.

Based on the above subgraph (subtree) extraction mechanism, we next describe our explainability technique, as shown in Figure 4.5, which can be integrated into any learning-based cost model. During the cost model's training process, the explainability technique will automatically learn to generate an explanation for subtrees in the query plan and identify a subset of nodes that are most influential for the cost model's prediction.

Given a tree model ϕ, a complete query plan (Original tree) G_{ot} and its node feature set X_{ot}, the technique's goal is to identify all K subtrees $G_{st} \subset G_{ot}$ and to evaluate the

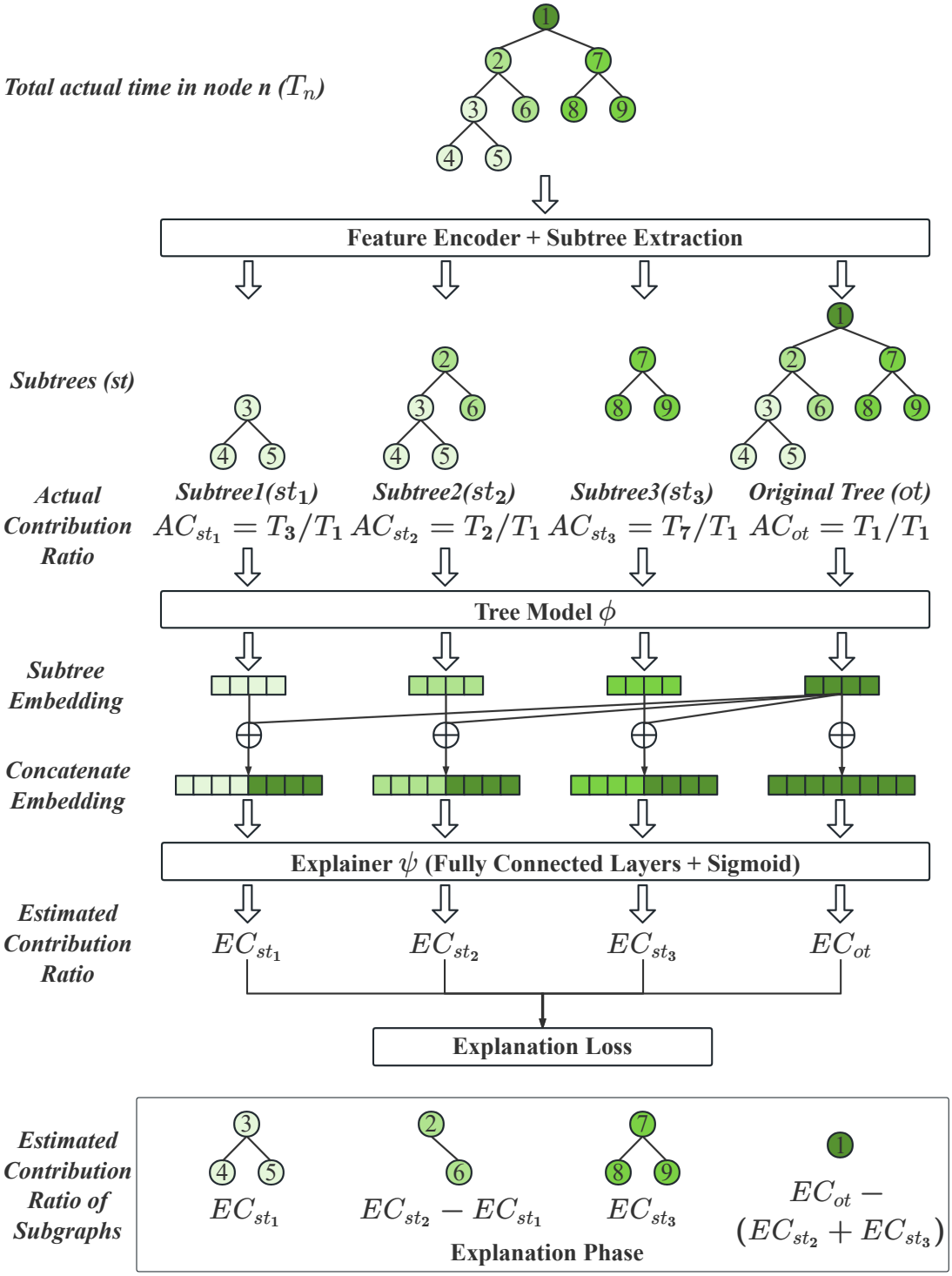


Figure 4.5: An example of the explainability technique for query plan cost model based on query plan subtrees.

contribution of each subtree to the final prediction with the subtree associated node feature set X_{st}. After extracting all subtrees, for each subtree G_{st_k}, the graph-level embedding Emb_{st_k} of the subtree is generated through the tree model ϕ:

$$Emb_{st_k} = \phi(G_{st_k}), k \in \{1, 2, ..., K\} \tag{4.7}$$

Similarly, for the original tree G_{ot}, which represents the complete query plan, we also use the tree model to generate its corresponding graph-level embedding, which is also the query plan representation used for the following cost estimation:

$$Emb_{ot} = \phi(G_{ot}) \tag{4.8}$$

Unlike traditional GNN explainers that utilize mutual information to quantify the impact of subgraphs by measuring changes in the probability of the final prediction, we develop a learning-based explainer model, denoted as Ψ, which is designed to estimate the influence of a subtree to the final cost prediction of the complete query plan. The model comprises multiple Fully Connected (FC) layers and uses a Sigmoid activation function. We employ the explainer model to estimate the correlation between two embeddings, specifically the embeddings of subtrees and the complete query plan. Due to the Sigmoid activation function, this correlation is always in [0, 1] and can be used to assess the contribution ratio that each subtree makes to the final prediction of the query plan cost. Thus, we first concatenate all embeddings of each subtree Emb_{st_k} to the embedding of original query plan Emb_{ot} respectively. The obtained concatenated embeddings are used as the input to the learning-based explainer Ψ and generate the estimated contribution ratio EC_{st_k} representing the contribution of the subtree k to the query plan cost prediction:

$$EC_{st_k} = \Psi(CONCATENATE(Emb_{st_k}, Emb_{ot})) \tag{4.9}$$

For the original tree, its concatenated embedding is composed of its own embedding combined with its embedding. Its Estimated Contribution Ratio EC_{ot} is shown as:

$$EC_{ot} = \Psi(CONCATENATE(Emb_{ot}, Emb_{ot})) \tag{4.10}$$

Since the explainer model operates using supervised learning, labels are required for the learning of contribution ratio estimation. As discussed in Section 4.2.1, each node within a query plan in workloads logs the cumulative cost for all nodes in the subtree for which it serves as the root, known as the total execution time (ET). Therefore, the actual execution cost for each extracted subtree is the total execution cost recorded in its root node. Given that predicting query plan costs essentially constitutes a regression problem, the contribution ratio of each subtree to the final prediction is determined by the ratio of that subtree's total execution cost to the total execution cost of the entire query plan, which is also in [0,1] because the execution cost of the theoretical query plan subtree cannot be longer than the execution cost of the complete query plan. Thus, the actual contribution ratio of each subtree is defined as:

$$AC_{st_k} = \frac{ET_{st_k}}{ET_{ot}}, AC_{st_k} \in [0, 1] \tag{4.11}$$

Where the actual contribution ratio of the original query plan tree is always 1. At this stage, we can utilize an explanation loss function to minimize the discrepancy between the estimated contribution ratio and the actual contribution ratio. Given a query plan set P with N plans, a plan p_i with K_i subtrees where $p_i \in P$, the explanation loss is shown as:

$$\text{ExplanationLoss} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\sum_{k=1}^{K_i}((AC_{st_{ik}} - EC_{st_{ik}})^2) + (1 - EC_{ot_i})^2}{K_i + 1}\right) \tag{4.12}$$

In this manner, the explainer automatically learns the relationship between the embedding of the subtree and the embedding of the complete query plan based on the tree model's output, thereby estimating the contribution ratio and explaining the query plan cost model. After model training is complete, for cost estimations that do not require explanations, Emb_{ot} can directly represent the query plan and perform cost estimations without needing to extract subtrees or activate the explainer, thus reducing the inference time of the cost model. When an explanation of the decision-making process for the predicted cost of a specific query plan is required, the explainability technique can be activated to estimate the contribution ratio of all subtrees to the final predicted cost. Utilizing the estimated contributions of all subtrees, we can explain the impact of each node's operation on the final prediction result, as outlined in Algorithm 2.

Algorithm 2 Explain the contribution for each query plan node operation based on subtrees

Input: Query plan tree T with root node r
Input: Estimated contribution ratios $EC_{\text{st}}(n)$ of the subtree rooted at each node n
Output: Estimated contribution ratios $EC_{\text{op}}(n)$ for each node's operation

```
function ComputeEC(n)
    if n is a leaf node then
        EC_op(n) ← EC_st(n)
    else
        for each child c of n do
            ComputeEC(c)
        end for
        EC_op(n) ← EC_st(n) − Σ_{c∈Children(n)} EC_st(c)
    end if
end function
ComputeEC(r) //Start explanation from the root node
```

Figure 4.5 demonstrates the architecture of our explainability technique for a learning-based cost model, where a complete query plan is inputted and extracted into various subtrees without destroying the structural integrity of query plans based on the subgraph extraction strategy. Each of these subtrees, such as those rooted at nodes 1, 2, and 7 called st_1, st_2, st_3, is independently encoded and processed to generate their plan-level embedding by the feature encoder and tree model ϕ in the cost model. The subtree's corresponding actual contribution ratio is calculated by dividing the total cost of each subtree by the total

cost of the complete query plan, which is used as the explanation label. After getting all the embeddings of the subtrees, these embeddings are concatenated with the embedding of the original tree, which is the complete query plan (ot). These combined embeddings serve as inputs to a learning-based explainer model Ψ, consisting of fully connected layers with a sigmoid activation function, which estimates the contribution ratios EC_{st} of these subtrees.

This process quantifies the influence of each subtree during the model's training phase, comparing the estimated contribution ratios EC_{st} against the actual contribution ratios AC_{st} of the subtrees to calculate an explanation loss. This evaluation assesses the accuracy of the explanation and the explainer model's capability to capture the relationship between the embeddings of the subgraph and the complete query plan. Furthermore, EC_{st} is utilized in the explanation phase to explain the specific influence of subgraphs on the final cost prediction. Notably, these subgraphs need not strictly be subtrees; the contribution of any specific subgraph can be deduced from the containment relationships between extracted subtrees. For instance, as shown at the bottom of Figure 4.5, if a subgraph exactly matches a subtree, its estimated contribution is directly output by the explainer model. Conversely, if the subgraph does not form a subtree, we can explain the irregular subgraph by subtracting the sum of the EC of the remaining subtrees except the required subgraph's nodes from the EC of the larger subtree that contains the subgraph. For example, the second subgraph in the bottom block is not a subtree and only includes nodes 2 and 6 but excludes nodes 3, 4, and 5, where the excluded nodes precisely match st_1, thus its contribution is inferred by subtracting the EC_{st_1} from EC_{st_2}. This method is similarly applied when explaining the contribution of the fourth subgraph with only single node 1, by subtracting the sum of EC_{st_2} and EC_{st_3}, which is the total EC of subtrees rooted at node 1's child nodes (nodes 2 and 7), from the EC of the subtree rooted at node 1.

Therefore, this technique allows for explaining each node's contribution to the final cost prediction. By aggregating or ranking these nodes' contributions, we can identify any nodes or subgraphs (not necessarily subtrees) within the query plan that most significantly impact the prediction results. At the same time, this mechanism enables the model to learn the execution information of intermediate nodes within the query plan from the workloads during the training process, which not only fully leverages the information involved within the query plan but also effectively augments the workload data, thereby reducing the number of samples required. This approach enables the learning-based cost model to function similarly to traditional cost models by demonstrating how each part of the query plan influences the final cost estimation. Consequently, database users and researchers can understand the decision-making process of the learning-based model and make precise, targeted adjustments.

4.3 A Robust Learning-to-Rank Cost Model Based on Uncertainty Quantification

Learning-based cost estimation has a probabilistic nature and inherently involves a degree of uncertainty [58]. A candidate query plan that has the lowest estimated cost but carries high uncertainty might not necessarily outperform another plan with a slightly higher cost estimate but lower uncertainty. This highlights the importance of considering both cost and uncertainty when evaluating the effectiveness of query execution plans. Therefore, exploring ways to either mitigate or leverage this uncertainty presents a promising direction for developing more robust cost estimation models. In this section, we describe our proposed learning-based cost model architecture, which is designed to quantify and integrate uncertainty into cost estimation and employs a ranking-based approach to plan comparison. This architecture enhances the ability to capture the potential variance in the cost model and apply it in plan selection, thus allowing for more informed and robust decision-making in query optimization.

4.3.1 Uncertainty Quantification

A robust plan is defined as one characterized by minimal degradation in worst-case performance in query optimization [46]. This attribute is quantified using the maximum Suboptimality and is suggested as a measure of robustness [58], defined as follows:

Suboptimality of a query plan p_i is the ratio of its execution cost $ET(.)$ to the cost of the actual optimal plan p_o and ranges from $[1, \infty]$:

$$Subopt(p_i) = \frac{ET(p_i)}{ET(p_0)} \tag{4.13}$$

Due to the existence of uncertainty in the query plan structure and cost model [58], the suboptimality of cost estimation during plan selection is significantly affected by the inaccurate estimated cost caused by the uncertainty. Figure 4.6 illustrates scenarios where the query plan cost estimate is inaccurate. Each subplot displays the estimated cost via the probability density function (PDFs) [104] for two alternative plans. The estimated costs are assumed to follow a normal distribution. In each scenario, we assume that the optimizer estimates the most likely cost as the expected cost for each plan. Due to the inherent probabilistic nature of cost estimations, estimated costs can vary, often falling within a predictable range influenced by uncertainties in these estimates. Lower uncertainty in a cost estimate indicates a higher likelihood that the estimated cost aligns closely with the expected cost. Conversely, higher uncertainty reduces the likelihood of selecting a plan that accurately reflects the expected costs, potentially leading to more inaccurate cost estimation. The plan with a lower expected cost is considered the optimal plan, while the other plan with a higher expected cost is the alternative plan.

Figure 4.6(a) illustrates the scenario where the query optimizer generates consistently accurate cost estimates for both plans, which are less affected by uncertainty. In such

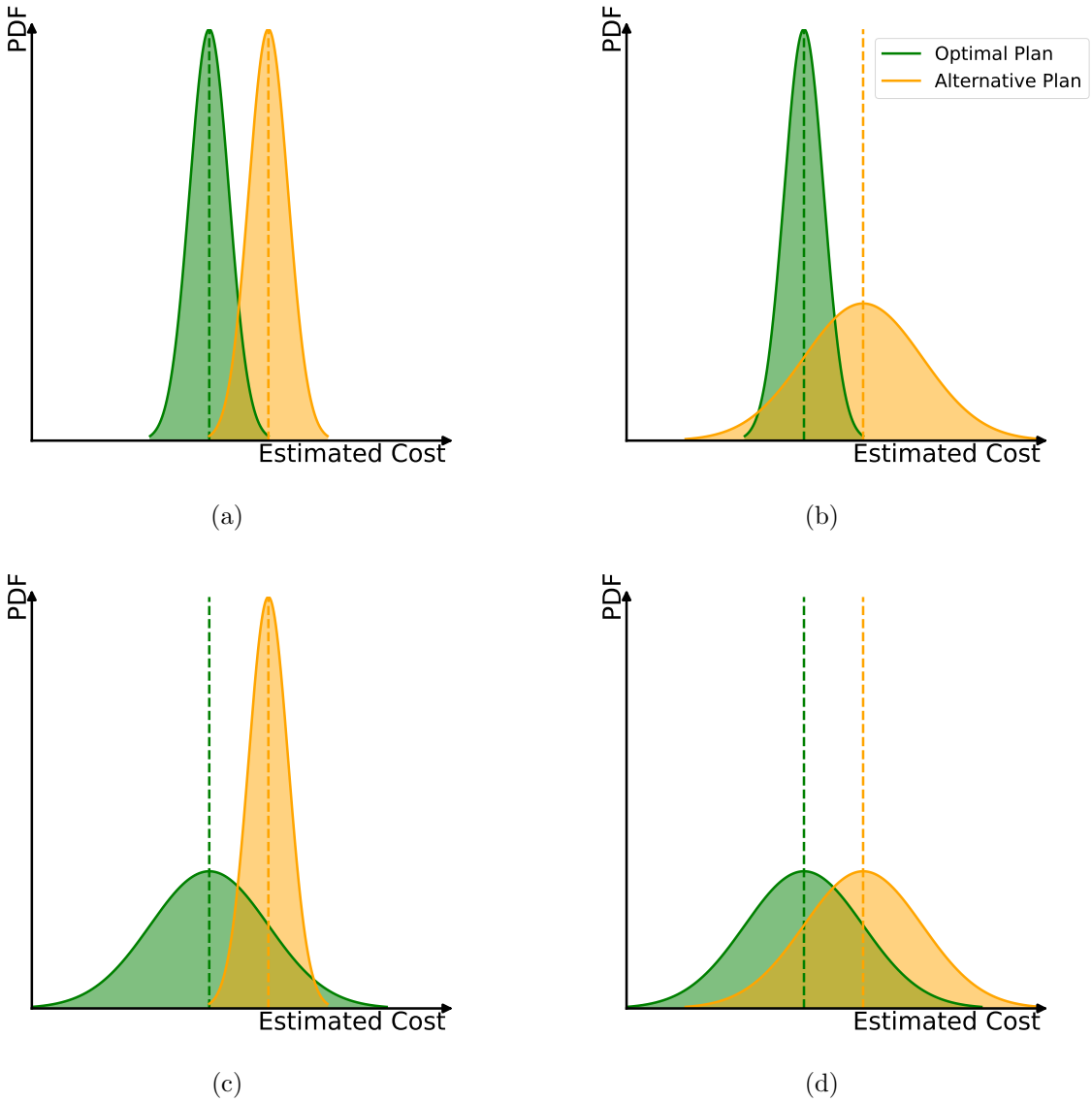


Figure 4.6: Scenarios for plan cost inaccuracies between an expected optimal plan and an alternative (second-best) plan.

a case, selecting the optimal plan is more likely to have a lower actual cost and result in better performance in terms of suboptimality. Conversely, Figures 4.6(b), 4.6(c) and 4.6(d) show scenarios with various levels of inaccurate cost estimations. Figure 4.6(b) and 4.6(c) displays a scenario in which the optimizer inaccurately estimates the cost of one of the two plans, substantially increasing the risk of selecting the more expensive plan. The worst-case scenario, presented in Figure 4.6(d), is the case when the optimizer inaccurately estimates both plans. This case further increases the risk, which makes it challenging for the optimizer to confidently select the more cost-effective option. These scenarios demonstrate the critical need to consider uncertainty in cost estimations rather than merely compare expected costs. By quantifying this uncertainty, the optimizer can make more informed decisions, leading to selecting a candidate plan with more likely lower estimated costs and

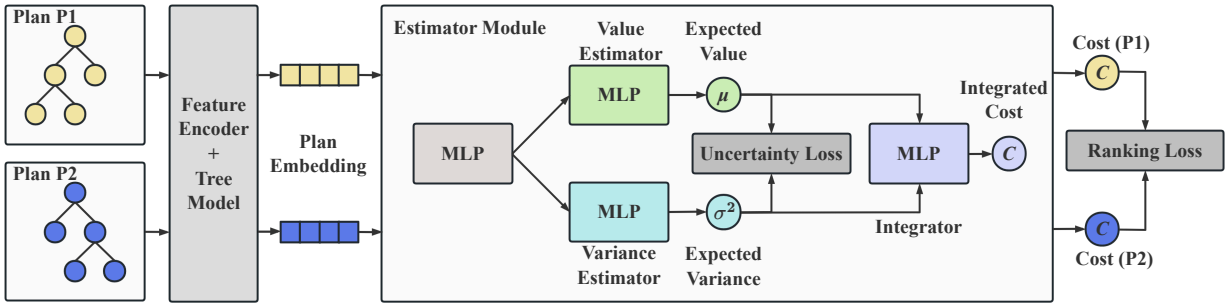


Figure 4.7: The architecture of the learning-to-rank robust cost model.

thereby ensuring more robust performance.

The variability inherent in real-world query plans can often be attributed to data uncertainty. In machine learning, this type of uncertainty can be caused by notice in input features or labels, or from input vectors that are too low-dimensional to adequately explain the sample. During the cost estimation process, uncertainty can arise from various complex factors. This includes fluctuations in execution time due to changes in the load or hardware conditions of the DBMS server when the sample query plan is executing, and variability or noise in the plan embeddings produced by the learning-based feature encoder and tree model from the query plan. Therefore, in this study, we focus on this uncertainty derived from data and develop a method to quantify it.

According to previous work, a neural network can be designed to predict the parameters of the normal distribution [100], which means that the model can automatically learn to predict not only the conditional expected value but also the conditional variance of the target based on the training data and the input sample. This functionality is achieved by integrating a secondary output branch into the original learning-based cost estimator that is tasked with variance prediction, as shown in Figure 4.7. The optimization of this model involves maximizing the log-likelihood with a Gaussian prior [100], as demonstrated in the following uncertainty loss function.

$$UncertaintyLoss = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\log \sigma_{p_i}^2}{2} + \frac{(y_{p_i} - \mu_{p_i})^2}{\sigma_{p_i}^2}\right) \tag{4.14}$$

Where for the embedding of i-th plan as input, μ_{p_i} is predicted by the first branch of the estimator module and represents the expected value of the estimated cost, $\sigma_{p_i}^2$ is from the second branch and represents the expected variance as the data uncertainty, and y_{p_i} stands for the label (actual cost) of the input plan. Based on this loss function, we can obtain both the estimated cost and the expected conditional variance for a given plan embedding. Utilizing these outputs, we can explain the plan's variability under different conditions, which allows us to evaluate the plan's uncertainty based on them effectively and incorporate this uncertainty assessment into the following plan selection phase.

4.3.2 Learning-to-Rank Pairwise Plan Comparison

As we discussed in Chapter 3, existing cost models that can quantify uncertainty usually apply the obtained uncertainty and estimated costs to the plan selection strategy independently of the model training phase. This separation prevents the model from optimizing itself based on the outcomes of plan selection. To address this limitation, we propose a novel learning-based cost model architecture as shown in Figure 4.7, which integrates uncertainty and cost estimates automatically and uses a ranking loss function with plan pairs as inputs, allowing the model to adaptively refine the integration strategy and improve the ability to compare and distinguish the actual performance of different plans, thereby providing a learnable robust plan selection and enhancing overall model performance.

Integration model. Our model integrates the estimated mean μ and variance σ^2 from the estimator module to compute the integrated cost C of a query plan p. The integration is performed using the equation below:

$$C = \alpha\left(W\begin{bmatrix}\mu\\ \sigma^2\end{bmatrix} + b\right) \tag{4.15}$$

Here, W is weight matrices, b is bias vectors, and α is an activation function (e.g., Sigmoid). This approach allows the model to optimize the integration process based on the characteristics of specific workloads.

Pairwise Plan Comparison. In addition to learning based on prediction cost accuracy, our cost estimator module also learns from pairwise comparisons of plans with binary labels indicating which one is better for each pair. Given a query plan set P with N plans, a pair of plans (p_i, p_j) where $\{p_i, p_j\} \in P$, and their estimated integrated cost (C_{p_i}, C_{p_j}) and their actual execution cost (y_{p_i}, y_{p_j}), a specified designed margin ranking loss function is used to enable the model to learn from comparative results, as illustrated below::

$$\begin{aligned} RankingLoss = \sum_{i=1}^{N}\sum_{j=i+1}^{N} \exp(\max(0, -y_{p_{ij}} \cdot (C_{p_i} - C_{p_j}) + \text{margin})) \\ \text{where } y_{p_{ij}} = \begin{cases} 1, & y_{p_i} > y_{p_j} \\ -1, & y_{p_i} \leq y_{p_j} \end{cases} \end{aligned} \tag{4.16}$$

Therefore, the complete loss function of this cost model should be:

$$Loss = UncertaintyLoss(Equation\ 4.14) + RankingLoss(Equation\ 4.16) \tag{4.17}$$

Through this architecture, the cost model can not only quantify uncertainty and estimate costs but also compare existing plans based on the integrated values of these two factors, enabling it to adaptively learn how to integrate the estimated uncertainty and cost. Conventionally, plan selection ranks all candidate plans by estimated cost and choosing the least expensive, making the decision highly dependent on the accuracy of these cost

estimates. However, the numerical precision of estimates based solely on cost is susceptible to various factors and uncertainties. Our approach, which utilizes ranking loss for pairwise plan comparison, enhances the model's capability to directly determine which of two plans is preferable by considering uncertainty rather than relying solely on numerical cost estimates. This method significantly bolsters the model's performance and robustness across multiple dimensions.

4.3.3 Uncertainty-aware Plan Selection

During the testing, our estimator model generates expected values μ and variances σ^2 for each query plan, which are integrated to compute the integrated cost as detailed in Equation 4.15. Plan selection is conducted based on the integrated cost and does the uncertainty-aware ranking. The specific plan selection strategy is outlined in Algorithm 3. It considers a candidate query plan set $P = \{p_i \mid i \in \{1, \ldots, N\}\}$ comprising N plans, where each plan p_i is associated with an integrated cost C_{p_i} derived from our integrator. Instead of merely selecting the plan with the lowest expected cost μ_{p_i}, the algorithm opts for the plan with the minimal integrated cost C_{p_i}, effectively balancing cost and uncertainty.

In addition, this architecture design offers an advantage over other ranking-to-learn cost models, such as Lero [182] and Leon [15], which do not support numerical cost estimation and instead categorize paired query plans to determine the better plan. Such models necessitate pairing various plans within the candidate set and conducting multiple comparison rounds to identify the optimal plan during plan selection. In contrast, our method benefits from the ranking-to-learn mechanism during training with paired plans but eliminates the need for pairwise comparisons during testing. Instead, plans are directly ranked based on their integrated costs, significantly cutting down on inference time.

Algorithm 3 Uncertainty-aware Plan Selection Algorithm

Input:
$P = \{p_i \mid i \in \{1, \ldots, N\}\}$
// The set of candidate plans in the search space
$C = \{(C_{p_i} \sim \mathcal{N}(\mu_{p_i}, \sigma^2_{p_i})) \mid i \in \{1, \ldots, N\}\}$
// Estimated cost and related variance for each plan
W, weight matrix; b, bias vector; Activation function α
Output:
p_{opt}, the selected optimal plan
1: Initialize an array CC to store computed conservative costs
2: **for** $i = 1$ to N **do**
3: $\quad CC_i \leftarrow \alpha\left(W \begin{bmatrix} \mu_{p_i} \\ \sigma^2_{p_i} \end{bmatrix} + b\right)$
4: $\quad CC \leftarrow CC \cup \{CC_i\}$
5: **end for**
6: $y := \arg\min_{i \in \{1,\ldots,N\}} CC$
7: $p_{\text{opt}} := P[y]$
8: **return** p_{opt}

Chapter 5

Model Architecture

By integrating the three techniques mentioned in Chapter 4, we propose a novel learning-based cost model that is designed to be robust to changes in input query plan samples and improves the accuracy of query plan execution time predictions while its decision-making process can be accurately explained. Figure 5.2 shows the architecture of the model Reqo. The model takes the query plan as input and outputs the integrated value of the expected execution cost and variables. The model can also learn directly from the comparison results of the different query plans. The architecture consists of a plan feature encoder and three main modules: the learning-based feature encoder can extract and encode the information of query plan nodes into fixed-length node features; the first module learns the representation of query plans; the second takes this representation to assess its impact on the integrated value of execution times and expected variance; and the third module utilizes subtrees from the input plan and learns to explain the decision-making processes of the first two modules concerning predicted execution times. In this chapter, we describe each component of architecture in detail. Section 5.1 shows the query plan feature encoder we constructed. Sections 5.2 to 5.4 present the implementation details of the representation learning, estimation, and explanation modules in our proposed model architecture, respectively. Section 5.5 describes the training methodology of the model.

5.1 Plan Feature Encoding

A query execution plan consists of information on the operators used to access and join data from various sources, including the physical implementation of the operators, their sequence, and the tables and columns accessed at each step. The amount of data flowing through each plan node, determined by the cardinality, is a decisive factor in the execution cost. In addition, the local and join predicates involved in the query plan may exhibit various levels of skewness and pairwise correlations, which in turn impact the accuracy of the cardinality estimates. To transform this complex information into fixed-length node features as input to the tree model, we used a plan encoder inspired by RTOS [163] while making critical changes to suit our experimental purposes. Unlike traditional learning-based cost model feature encoders that rely on estimated cardinality or cost from traditional

DBMS as part of node features, our encoder learns directly from the query plan. This approach ensures that our model's feature encoding remains unaffected by the inherent limitations and performance variability of traditional cost models. As shown in Fig. 5.1, each node feature in the query execution plan tree is composed of three parts: Node Type Embedding, Table Embedding, and Predicate Embedding.

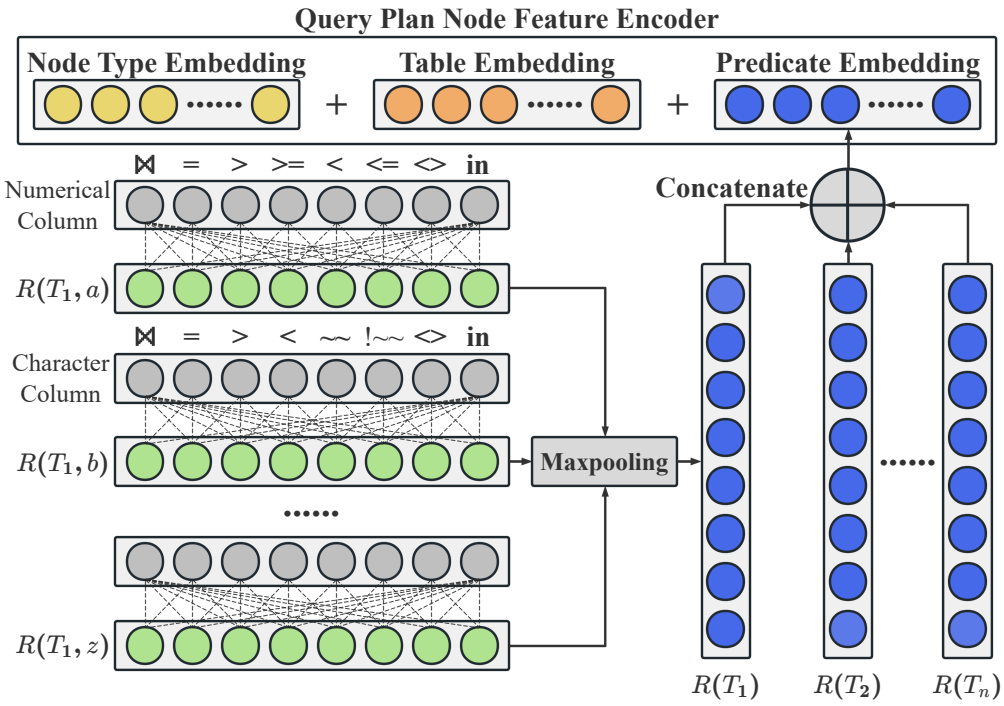


Figure 5.1: Learning-based query plan node feature encoding.

Node Type Embedding. The node contains a specific operator type, such as Hash Join or Index Scan. We perform one-hot encoding on the node types and pass through a Fully Connected (FC) layer to obtain the node type embedding $E(nodetype)$.

Table Embedding. We apply one-hot encoding on all tables used in the node's operation to obtain the Table Embedding $E(table)$, ensuring that no information is lost concerning nodes that perform operations based on a table but no predicates related to that table.

Predicate Embedding. We consider the two main types of columns involved in the predicate:

For numerical columns with values, the predicate operations in the node can be classified into 8 cases: $\bowtie$, $=$, $>$, $\geq$, $<$, $\leq$, $\neq$, in. For a column c, each c is represented by a feature vector $F(c)$ of the same length 8:

$$F(c) = (c_{\bowtie}, c_{=}, c_{>}, c_{\geq}, c_{<}, c_{\leq}, c_{\neq}, c_{\text{in}}) \tag{5.1}$$

If the predicate involves a specific operation on a certain column, the corresponding vector will be encoded according to the specific predicate type. If c exists in a join predicate,

the corresponding position $c_{\bowtie} = 1$ otherwise 0. For the other five cases, the predicate value v is normalized to $v_{nor} \in [0, 1]$ using the maximum (c_{max}) and minimum (c_{min}) values of the column in the database and plus 1 as the value of the corresponding position. If the predicate value exceeds the value range of the column in the database, according to the operator type, if there is no tuple that satisfies the predicate, the corresponding position is set to -1; otherwise, 2. Uninvolved cases remain at 0. The encoding details of the five cases are shown below:

- For predicate $c = v$ or $c \neq v$, if $v < c_{min}$ or $v > c_{max}$, we set $c_{=} = -1$ because we cannot retrieve any data in that situation, otherwise $c_{=} = v_{nor} + 1 = \frac{v - c_{min}}{c_{max} - c_{min}} + 1$.
- For predicate $c > v$, if $v \geq c_{max}$ then $c_{>} = -1$, else if $v < c_{min}$ then $c_{>} = 2$, otherwise $c_{>} = 2 - v_{nor}$.
- For predicate $c \geq v$, if $v > c_{max}$ then $c_{\geq} = -1$, else if $v \leq c_{min}$ then $c_{\geq} = 2$, otherwise $c_{\geq} = 2 - v_{nor}$.
- For predicate $c < v$, if $v \leq c_{min}$ then $c_{<} = -1$, else if $v > c_{max}$ then $c_{<} = 2$, otherwise $c_{<} = 1 + v_{nor}$.
- For predicate $c \leq v$, if $v < c_{min}$ then $c_{\leq} = -1$, else if $v \geq c_{max}$ then $c_{\leq} = 2$, otherwise $c_{\leq} = 1 + v_{nor}$.
- For predicate c in $\{v_1, v_2, \ldots, v_k\}$, $n_v = \sum_{i=1}^{k} \mathbf{1}$ if $c_{\min} \leq v_i \leq c_{\max}$ then $c_{\text{in}} = 1 + \frac{n_v}{\text{count}_{\text{unique}}}$, where $\text{count}_{\text{unique}}$ denotes the number of unique values in the column.

For other value-type columns, such as string, it cannot be mapped to a value with interval meaning by a simple method (e.g. hash) [163]. Thus, the join predicate is encoded in the same way as above. For the other cases (=, >, <, like, not like, ≠, in), we use word2vec [94] to translate characters into numerical values for their respective positions in the column feature.

Each column has a dedicated matrix $M(c)$ with shape $(6, hs)$ for each column c to process its encoded feature vector and generate learned column embeddings $E(c)$ with length hs. The embedding $E(c)$ with shape $(1, hs)$ is given by:

$$E(c) = F(c) \times M(c) \tag{5.2}$$

Then, max pooling is performed on all column embeddings belonging to the same table to obtain the embedding expression $E(t)$ of each table with the same length. Since we need to avoid information loss when aggregating embeddings in the node feature encoding stage, we directly concatenate embeddings of all tables as the predicate embedding. More specifically, given a table with k columns, we concatenate the k column embeddings $(E(c_1), E_(c_2), \cdots, E(c_k))$ into a matric $M(t)$ with the shape (k, hs), $M(t) = (E(c_1) \oplus E(c_2) \oplus \cdots \oplus E(c_k))$, and apply an max pooling layer (kernel shape $(k, 1)$) to get the table embedding of shape $(1, hs)$, as:

$$E(t) = \text{MaxPool}(M(t)) \tag{5.3}$$

Next, give a workload with m tables, the m table embeddings $(E(t_1), E_(t_2), \cdots, E(t_m))$ are concatenated into the predicate embedding $E(predicate)$ with the shape $(1, hs \times m)$, $E(predicate) = (E(t_1) \oplus E(t_2) \oplus \cdots \oplus E(t_m)$. Then we can concatenate the obtained Node Type Embedding $E(nodetype)$, Table Type Embedding $E(table)$ and Predicate Embedding $E(predicate)$ to get the query plan's node feature:

$$Node\ Feature = E(nodetype) \oplus E(table) \oplus E(predicate) \tag{5.4}$$

5.2 Representation Learning Module

The representation learning module takes the information in the encoded query plan tree obtained after feature extraction as input and generates the corresponding plan-level embedding. This module serves a role similar to that of the tree model in the learning-based query optimizer framework. In this model, we use our proposed tree model, which incorporates 4 bidirectional GNN layers and a GRU aggregation layer as detailed in Section 4.1. This architecture handles a vectorized query plan tree as input, and each bidirectional GNN layer transforms the input plan tree with undirected edges into two tree graphs with unidirectional edges, each having the same nodes but in opposite directions (from parent node to child node and vice versa). The two graphs are processed by two independent TransformerConv [121] layers, respectively, after which corresponding node features in the two trees are weighted and aggregated back into a new undirected edge tree as the output. After multiple layers of bidirectional GNN, the vectorized tree is expanded into a node sequence through post-order traversal and is subsequently passed to a GRU-based GNN aggregation operator [9]. This step aggregates the node information to produce the final query plan representation. Our experiments confirm that this tree model architecture performs more effectively than other state-of-the-art tree model techniques.

5.3 Estimation Module

The estimation module takes the plan representation produced by the representation learning module as input. The specific principles and details of this module architecture are described in Section 4.7. This representation is first processed through a multi-layer perceptron (MLP), which consists of 3 fully connected layers. The output from this MLP module is then fed into two separate branches: each is also composed of MLP with 3 fully connected layers. The first branch employs a Sigmoid activation function to transform the MLP output into a normalized expected execution time for the plan. Concurrently, the second branch utilizes a SoftPlus function to transform its MLP output into a non-negative variance of the first branch's predicted execution time, representing uncertainty.

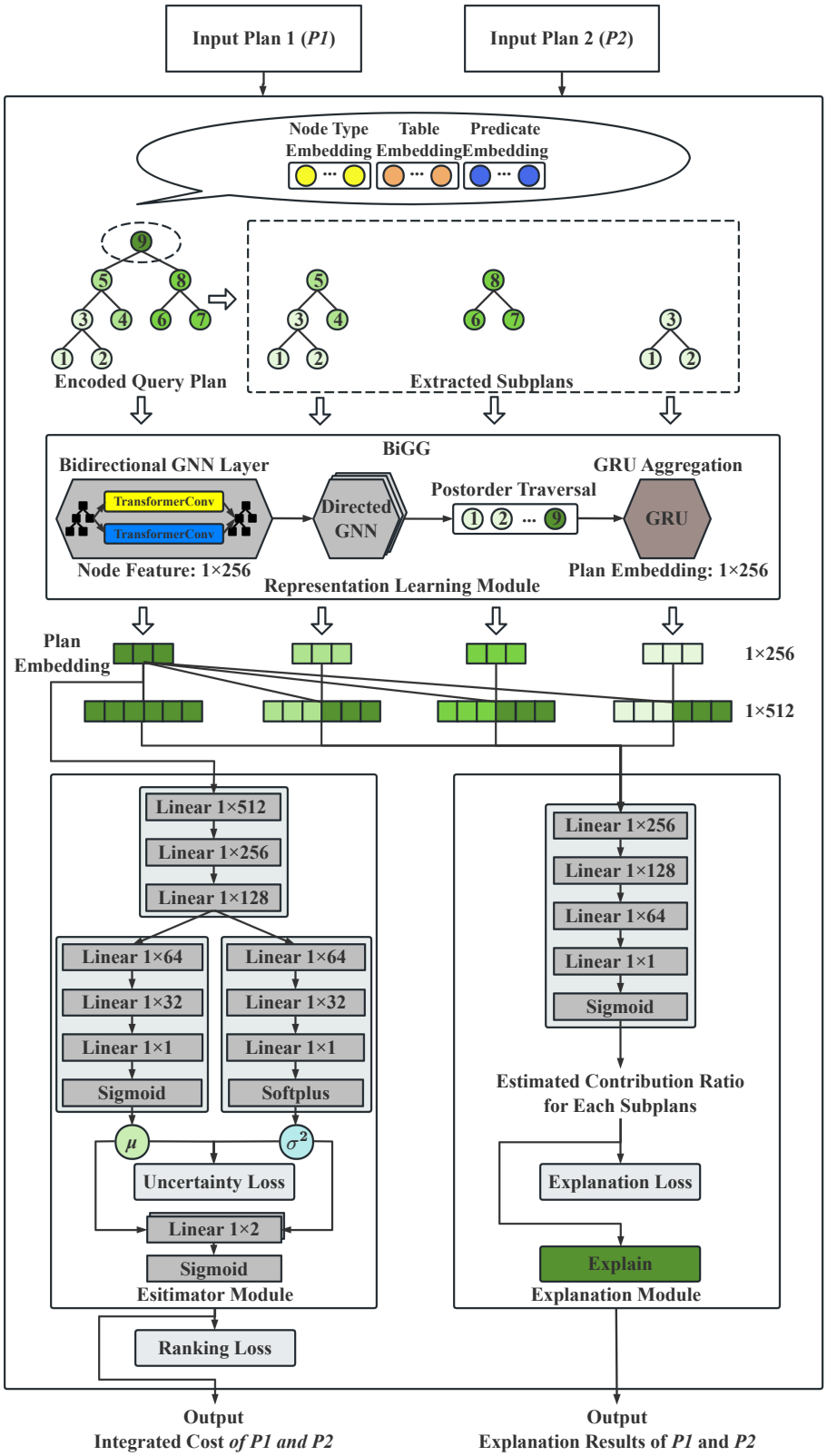


Figure 5.2: The complete architecture of our robust and explainable cost model.

The outputs of the two branches are subsequently integrated by an MLP-based model using 2 fully connected layers and a Sigmoid activation function, resulting in an integrated value. This integrated value is used for subsequent plan comparison or ranking, effectively leveraging both predicted execution times and their associated uncertainties to improve the plan selection.

5.4 Explanation Module

After being encoded by the feature encoder, the explanation module first extracts the subtrees from the vectorized query plan tree. This extraction is performed using the subtree extraction mechanism detailed in Section 4.2.1. The representation module then processes the subtrees to obtain their respective subplan-level embeddings. Then, we concatenate the subtree embeddings derived from the same plan and their corresponding complete plan embeddings. These concatenated embeddings are used as inputs to the explainer module, which is composed of MLP layers with 4 fully connected layers and a Sigmoid activation function to predict the contribution ratio of each subtree towards the predicted execution time of the entire plan. This ratio ranges from 0 to 1, allowing the module to deduce the estimated contribution value of each subgraph or node based on interrelationships between these subtrees. This method enhances the model's interpretability and credibility by providing explanations of the decision-making process behind the execution time predictions for the entire plan. Based on the model's architecture, the learning-based explanation module can also provide feedback and further improve the cost estimation performance of the model.

5.5 Model Training

The training of this model requires input in the form of query plan pairs, using the actual execution times of these plans and their subplans as labels. This is because our model is designed to compare uncertainty-incorporated execution time estimates between different plans and can leverage ranking loss 4.16 to automatically learn from these comparisons. Within the model's three main modules, the estimation module is trained using both uncertainty loss 4.14 and ranking loss. The explanation module is trained using its explanation loss.

The explanation module's involvement in the model architecture is optional. If the model does not require an explainability mechanism, it can be trained using the loss function specified in Equation 4.17. However, if the explainable mechanism is activated, the combined sum of the three loss functions from the model's architecture serves as the overall loss function, as indicated in Equation 5.5.

$$Loss = UncertaintyLoss(Eq.\ 4.14) + RankingLoss(Eq.\ 4.16) + ExplanationLoss(Eq.\ 4.12) \tag{5.5}$$

The model does not require inputs to be paired in practical application or during the testing phase. After training, plans can be directly ranked based on the expected execution time output and the integrated value produced by the estimation module with the learnable parameter, thereby achieving the prediction of execution times and plan selection incorporating uncertainty.

Chapter 6

Experimental Study

In this chapter, we conduct an experimental study aimed at comparing the performance of state-of-the-art learning-based tree models and GNN-based tree models, including our proposed BiGG tree model, across various task scenarios. This study also aims to explore the potential improvements in robustness and explainability in query optimation provided by our proposed robust cost model and explainability technique, comparing these enhancements with an advanced commercial query optimizer PostgreSQL and other state-of-the-srt learning-based cost models. Section 6.1 outlines the experimental setup and evaluation metrics. Section 6.2 presents a comparative study based on state-of-the-art tree models. Section 6.3 details our experimental study on the cost estimation, robustness and explainability of our proposed model Reqo and other advanced learning-based cost models.

6.1 Experimental Setup

The experiments are performed on a Linux server with an 8-core Intel Silver 4216 Cascade Lake 2.1GHz CPU, 128GB memory, and a 32GB NVIDIA V100 Volta GPU. PostgreSQL 15.1 is used as the RDBMS for compiling and executing the queries to generate workloads. Since our proposed cost model is not yet integrated into a complete query optimizer like PostgreSQL, it still relies on query plans generated by the PostgreSQL optimizer, as well as their PostgreSQL optimizer's estimated cost, to be used as baselines for evaluating the cost estimation and plan selection performance of models. We consider the plan with minimum execution time among all plans in the search space to be the optimal plan. The prototype code is written in Python 3.10 with the machine learning library PyTorch [105]. Hyperparameter tuning is performed using Ray Tune [77]. All experimental results are the average after 10-fold cross-validation.

Benchmarks. We evaluate all the query optimizer cost models on four widely used benchmarks.

1. The STATS dataset and STATS-CEB workload [45] have been recently introduced for assessing the end-to-end performance of query optimizers. The STATS dataset

comprises 8 tables of user-contributed content from the Stats Stack Exchange network, featuring a more complex data distribution compared to IMDB. STATS-CEB includes 146 query templates with varying join sizes and types. To enable large-scale evaluation in most of our experiments, we created a workload of 3,000 queries based on the templates. For each query, we randomly selected a query template from JOB-light, retrieved its join template, and then added randomly generated predicates.

2. The IMDB dataset consists of 21 tables related to movies and actors, with the JOB-light workload [70] including 70 real-world queries from 2 joins to 5 joins. Using the same method described earlier, we generated a workload of 2,600 queries.

3. The TPC-H benchmark [108] is a benchmark for testing the performance of databases with complex business queries and data modifications and has its data synthetically generated with a uniform distribution. For our experiments, we used TPC-H 3.0.1 to generate a 10 GB database consisting of 8 tables with 61 columns. Based on all 22 query templates, we generated a workload of 1,100 queries by varying the predicates under each template.

4. The TPC-DS [109] is another benchmark for evaluating database performance. We use workloads based on a 10 GB database generated by TPC-DS 3.2, an industrial standard benchmark commonly used to evaluate the performance of cost estimation with 25 tables and 429 columns. TPC-DS has more relations and allows for more complex query patterns compared with the above benchmarks. Therefore, using TPC-DS, we can generate more complex join queries to evaluate the representation learning performance of the tree models in more complex scenarios. Due to the limited complexity of the query generation template that comes with TPC-DS, we used a random query generator to generate queries based on TPC-DS relations. The queries are determined by parameters provided to the query generator, including the number of joins, join types (such as inner-join, outer-join, left/right outer-join, and anti-join), the number of local predicates, and the types of operators used in join and local predicates. In this experimental study, we utilize this query generator to generate a total of 22k queries, and each query has up to 10 joins, sampled from referential integrity one-to-many and artificial many-to-many joins, with each join having up to 3 join predicates and each table with up to 5 local predicates. We used these queries to generate workloads in PostgreSQL, serving as the experimental data.

We select and preprocess the necessary data from these workloads according to the specific requirements and characteristics of different task scenarios, thus forming the datasets for each specific experimental task. The following are the datasets used in our experiments for different task scenarios:

1. *Dataset for cost estimation accuracy.* We select PostgreSQL's default compiled query plans as samples based on the generated queries, using their execution times as labels.

2. *Dataset for plan selection and robustness performance.* Each generated query was compiled in PostgreSQL using 13 different hint sets inspired by Bao [88]. These hint

sets introduced specific constraints on the join and access operators utilized within the query plan generation. Each sample in the dataset is a collection of candidate query plans generated based on the 13 hints from the same query.

3. *Dataset for explainability performance.* For each of the generated queries, the workloads of PostgreSQL generated default query plan and all its subtrees extracted based on the strategies described in Section 4.2.1 are used as dataset samples. The total execution time of these complete plans and all sub-plans is collected to generate the explanation labels. To optimize resource usage and training speed in this experimental study, we have set the minimum subtree extraction size to at least two nodes, meaning individual leaves are not considered as subtrees.

Dataset Preprocessing. The collected labels in the above three datasets are preprocessed by applying natural log transformation and min-max scaling. To reduce the significant skewness in execution latency values, we first apply a natural log to all labels and make them more suitable targets for a machine learning model to learn [58], which is performed as the following formula, given a plan's actual execution time as label y, where $y_{\log_e}$ represents label after the natural logarithmic transformation.

To address significant skewness in execution time values from the workloads and optimize them for machine learning analysis, we preprocess the labels from the datasets as follows: A natural logarithmic transformation is first applied to all labels. This transformation is executed as follows:

$$y_{\log_e} = log_e(y) \tag{6.1}$$

where y represents a plan's actual execution time, and $y_{\log_e}$ is the label after the natural logarithmic transformation. Therefore, we can transform the highly skewed labels through log transformation and make them more suitable targets for a machine learning model to learn [58].

After the natural logarithmic transformation, min-max scaling is performed based on the processed label to normalize it to the range [0, 1] so that it is in the same range as the output of the Sigmoid activation function used in the final layer of the cost estimator. The preprocessed label after scaling y_p is defined as:

$$y_p = \frac{y_{\log_e} - \min(y_{\log_e})}{\max(y_{\log_e}) - \min(y_{\log_e})} \tag{6.2}$$

Where $\min(y_{\log_e})$ and $\max(y_{\log_e})$ are obtained from the labels in the training data and used for validation and testing data.

Evaluation Metrics. We employ 7 evaluation metrics for the experiments, as shown in the following. Metrics 1-3 are mainly applied to the cost estimation task, metrics 4-5 are dedicated to the plan selection and robustness task, while metrics 6-7 are designed for the cost estimation explanation task.

1. *Prediction Error:* We use Q-Error to evaluate the performance of cost estimation based on the magnification difference of predicted execution time and actual execution time. Given an output of the estimator module y_{out}, where is the output of the activation function Sigmoid in the range $[0, 1]$ and actual execution time y_{a}, the y_{out} needs to undergo the inverse process of natural logarithmic transformation and min-max scaling to obtain the predicted execution time y_e, which is as shown as follows:

$$y_e = e^{\left(y_{out} \times (\max(y_{\log_e}) - \min(y_{\log_e})) + \min(y_{\log_e})\right)} \tag{6.3}$$

   Then, the Q-Error is defined as:

$$Q\text{-}Error = \frac{\max(y_e, y)}{\min(y_e, y)} \tag{6.4}$$

2. *Correlation:* We use Spearman's rank correlation to measure the relationship between model-estimated latency and actual latency, with values closer to 1 indicating a stronger correlation. Unlike the widely used Pearson's coefficient, Spearman correlation is not such sensitive to outliers and data scales for using a monotonic function, making it more suitable to evaluate indicators such as latency that may have large orders-of-magnitude differences. The formula for Spearman's rank correlation is defined as:

$$\rho = 1 - \frac{6 \sum_{i=1}^{N} (r_{y_e,i} - r_{y,i})^2}{N(N^2 - 1)} \tag{6.5}$$

   Where $r_{y_e,i}$ and $r_{y,i}$ represent the ranks of the predicted execution time y_e and the actual execution time y, respectively, and N is the number of the test set.

3. *Inference Overheads:* We measured the average time the tree model used to predict a latency estimate for each encoded query plan during testing, which can be used to compare the computational overhead of each tree model.

4. *Plan Suboptimality:* Given a set of candidate execution plans p generated based on the same query, ranking these candidate plans according to the actual execution time $ET(.)$ and identifying the plan with the shortest execution time as the actual optimal plan, denoted as p_o. Concurrently, given a plan $p_i \in p$, we define its plan suboptimality as the ratio of its execution cost to the cost of the optimal plan p_o and takes a value in the range of $[1, \infty)$:

$$\text{Plan Suboptimality} = \frac{ET(p_i)}{ET(p_o)} \tag{6.6}$$

   Plan suboptimality reflects the model's ability to identify the optimal plan among multiple options to some extent. By analyzing this metric, we can assess the performance distribution of the plans selected by the model, particularly focusing on

the tail of the distribution, which highlights the model's performance degradation in worst-case scenarios. This assessment allows us to understand the model's specific performance when selecting different plans, serving as a measure of its robustness.

5. *Total Runtime:* The total running time ratio is defined as the ratio of the total actual execution time of the plan selected by the model to the total execution time of the actual optimal plan (or the plan selected by PostgreSQL) for all queries. Compared with Plan Suboptimality, this metric is less affected by outliers and can capture a major slowdown for the given workload, providing a more stable and comprehensive view for evaluating and comparing the performance of different tree models. The total running time ratio is given by:

$$\text{Total Runtime} = \frac{\sum_{i=1}^{N} ET_m(q_i)}{\sum_{i=1}^{N} ET_{opt/postgres}(q_i)} \tag{6.7}$$

Where $ET_m(q_i)$ is the actual execution time of the plan selected by the model for query q_i, $ET_{opt/postgres}(q_i)$ is the execution time of the actual optimal plan or the PostgreSQL-selected plan for query q_i, and N is the total number of queries.

6. *Explanation Accuracy Ratio:* We define the explanation accuracy ratio as the ability of the model or the explainability mechanism to accurately identify the subgraph(s) that contribute most significantly to the final prediction. This metric is calculated by determining the percentage of samples for which the model correctly identifies the most influential subgraph(s), out of the total number of samples tested. This metric yields a value in the range of $[0, 1]$, where the value closer to 1 indicates higher explanation accuracy. This metric allows us to assess the model's performance in terms of explainability. The explanation accuracy ratio is given by:

$$\text{Explanation Accuracy Ratio} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(S_{pred,i} = S_{true,i}) \tag{6.8}$$

Where $S_{pred,i}$ is the set of subgraphs identified by the model in descending order of contribution for query plan i, and $S_{true,i}$ is the actual ordered set of most influential subgraphs in descending order for that plan. The indicator function $\mathbb{I}$ returns 1 if both the elements and the order of the subgraphs in the sets are identical, and 0 otherwise. N represents the total number of query plans. This metric yields a value in the range of $[0, 1]$, with values closer to 1 indicating higher explanation accuracy. By varying the number of top-ranked subgraphs used as the criterion for explanation accuracy, we can evaluate the model's performance under different scenarios, including the model's ability to identify the most critical subgraph or find most of the important subgraphs in a more comprehensive way, which provides a comprehensive understanding of the model's explanatory capabilities across different interpretative demands.

7. *Explanation Subgraph Influence Ratio:* Similar to plan suboptimality, consider all the subgraphs S extracted from a query plan based on a specific extraction strategy.

These subgraphs are ranked in two ways: reverse-ranked by their estimated contribution, denoted as $EC(\cdot)$, forming the set S_e, and reverse-ranked by their actual contribution, denoted as $AC(\cdot)$, forming the set S_a. Select the top K most influential subgraphs identified by the model from S_e, denoted as $s_{e_k} \in S_e$, and the top K actual most influential subgraphs from S_a, denoted as $s_{a_k} \in S_a$, where $k \in \{1, 2, \ldots, K\}$. The explanation subgraph influence ratio, considering multiple top-ranked subgraphs, is defined as:

$$\text{Expl Subgraph Infl Ratio} = \frac{\sum_{k=1}^{K} AC(s_{e_k})}{\sum_{k=1}^{K} AC(s_{a_k})} \tag{6.9}$$

This metric evaluates the model's capability to identify the most influential subgraphs. Unlike the explanation accuracy ratio, which measures whether the model correctly identifies the single most important subgraph, the influence ratio reflects the cumulative contribution of the subgraphs selected by the model relative to the actual top-performing subgraphs. This provides a more nuanced assessment of the model's explainability by focusing on the proportional influence of its selections.

Model Training and Parameter Tuning. In the tree model comparison, during the model training phase, the hyper-parameters are individually tuned for each tree model. The learning rate and other hyper-parameters of the remaining modules, such as the feature encoder or cost estimator, are jointly tuned. For the other experiments, the hyperparameters of each module in our proposed model architecture are tuned individually. Parameter tuning is done using the Asynchronous Successful Halving Algorithm (ASHA) [74], which allows for exploring a large number of parameter combinations while limiting the total amount of time needed for tuning. Adaptive Moment Estimation (ADAM) [63] is used as the optimizer for model training, which is a first-order gradient-based method that adjusts learning rates for individual parameters by calculating the exponentially weighted averages of past gradients (first moments) and squared gradients (second moments). Adam optimizes parameter updates by using these moment estimates to adaptively scale the learning rates, allowing for more controlled and efficient adjustments during training, especially in scenarios with large datasets and parameters, non-stationary objectives, and noisy or sparse gradients [63]. Furthermore, dropout and early stopping are used to avoid over-fitting.

6.2 Tree Model Comparison

Cost estimation is a crucial step in the query optimizer, as it directly determines how the model selects the optimal plan from candidate plans. However, these cost estimates are subject to arbitrary errors rooted in various sources, such as inaccurate cardinality estimates or simplifying assumptions. Therefore, in this work, we opt to learn the plan execution latency directly from runtime.

In addition to evaluating tree model cost estimation error and correlation, model performance in plan selection is also crucial. In the plan selection phase of query optimization,

due to data and model uncertainties [58], the model's cost predictions may have inevitable errors, potentially affecting the final optimal plan selection. In practical applications, if a model can select the actual optimal plan or not is far more important than making more accurate predictions. Therefore, we evaluate the model's ability to correctly select the optimal plan with the shortest actual execution latency among multiple candidate query plans. We use 13 candidate plans to simulate plan selection application scenarios to explore and compare the impact of employing different tree models on the overall plan selection accuracy of the optimizer.

To assess the impact of different tree models in query plan representation, we use the same feature encoding model described in Section 5.1 and the same MLP-based cost estimator for all experiments. In this way, we can evaluate the representation ability of different tree models through the accuracy of cost estimates and plan selection.

6.2.1 Experimental Methodology

Selected Existing Tree Models. We select five state-of-the-art tree models for further evaluation and comparison as follows:

1. *LSTM:* LSTM is a variant of recurrent neural network (RNN) architecture designed for sequences of data and capable of remembering long-term dependencies [125]. Since it cannot directly handle the tree, we flatten the query plan according to post-order traversal [83] as input. The hidden layer of the last node is treated as a graph-level representation of the query plan.

2. *GRU:* GRU is a type of RNN that simplifies the LSTM architecture by combining the forget and input gates into an update gate, improving efficiency and performance on tasks involving sequential data [18]. Model input and output are obtained in the same way as LSTM.

3. *LSTM + Self Attention:* Saturn [83] proposed an improvement of LSTM for query plan representation by applying the Self Attention [132] mechanism to weigh each hidden layer and aggregate all hidden layers according to their weight as the plan-level representation.

4. *Tree-LSTM:* Tree-LSTM is an adaptation of the conventional RNN for tree-structured graphs, which can aggregate information from the leaf nodes to the root across different tree branches by generalizing the traditional LSTM cell by accepting inputs from multiple channels [126]. The root node representation is treated as the graph-level representation of the query plan.

5. *Tree-CNN:* Tree-CNN is designed to handle tree-structured data [95], and was first applied to query plan representation in NEO [89]. It slides a triangular kernel from the root to the leaf nodes, which allows learning relationships between each combination of two child nodes and their parent node. Dynamic pooling is used to aggregate all features into a graph-level representation of the query plan [89]. This model can

Table 6.1: Cost Estimation Accuracy (Q-Error) of tree models in TPC-DS workload. The best-performing model for each metric is highlighted in **bold and underlined**.

Tree Model (+Aggregation Method)	Graph Edge Direction	Median Q-error	90th Q-error	99th Q-error	Top 50% Mean Q-error	Top 99% Mean Q-error	Spearman's Correlation
GRU	-	1.895	22.582	296.242	1.378	5.727	0.776
LSTM	-	1.954	21.979	311.205	1.398	5.824	0.765
LSTM + Self-Attention	-	1.885	20.710	313.775	1.370	5.489	0.778
Tree-LSTM	-	1.840	19.413	235.727	1.352	5.063	0.783
TCNN	-	1.912	25.758	430.101	1.380	6.674	0.761
GNN + AddPool	Single directed	1.971	25.184	471.242	1.403	6.697	0.765
GNN + GRU	Single directed	1.884	20.960	324.524	1.369	5.684	0.776
GNN + GRU	Undirected	1.880	20.320	298.524	1.371	5.409	0.775
Bidirectional GNN + AddPool	Weighted directed	1.882	23.387	459.604	1.370	6.502	0.771
Bidirectional GNN + GRU	Weighted directed	**1.762**	**16.537**	**199.137**	**1.327**	**4.558**	**0.805**

only be applied to binary trees. Non-binary trees require preprocessing, involving adding more nodes and layers, to become compatible with the model.

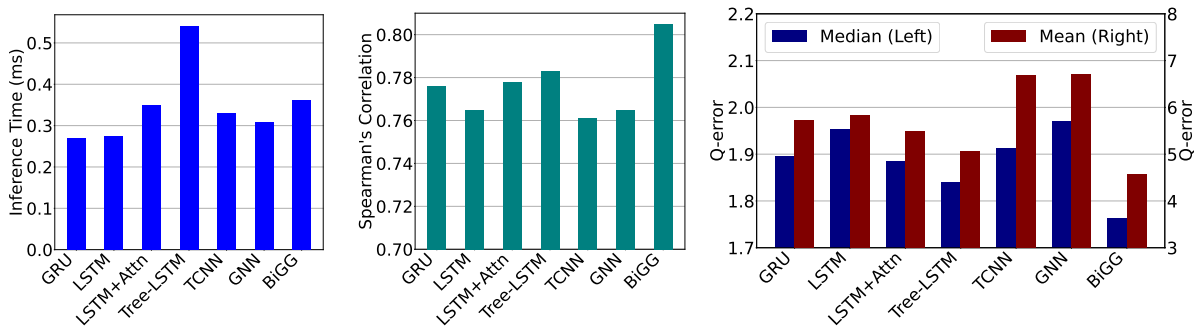
Loss Function. We used the Mean Squared Error algorithm based on the cost estimator model's output and labels after being preprocessed by natural log transformation and min-max scaling. Given all the labels y_p in the dataset after logarithmic transformation and min-max scaling, a cost estimator model output y_{out}, the loss function is defined as LogMSE and is shown as follows.

$$\text{LogMSE Loss} = \sum_{i=1}^{n} \left(y_{p_i} - y_{out_i}\right)^2 \tag{6.10}$$

6.2.2 Existing Tree Model Cost Estimation Performance and Analysis

We evaluate the impact of using different tree models under the same framework on the overall cost estimation performance of the query optimizer. We present the results of the current state-of-the-art tree models in the upper part of Table 6.1, where the first block includes the state-of-the-art tree models we evaluated. The second block includes the tree models based on GNN layers with different graph edge directions and aggregation methods. The overall best results across blocks are underlined. Part of the comparison results are demonstrated in Figure 6.1, where (a) is the inference time of each tree model, (b) is Spearman's rank correlation coefficient and (c) is the mean and median Q-error experimental results.

Even in workloads with larger scale and more complex join relationships, five existing tree models, including GRU, LSTM, LSTM + Self-Attention, Tree-LSTM, and TCNN, do not have a significant difference in query plan representation capabilities, as stated in the study by Yao Z. et al. [175]. However, in the special scenario of representing complex query plans, the differences in Q-Error and Spearman's Correlation performance between different models become more apparent. We outline interesting observations regarding the impacts of various characteristics of the models on plan tree representation learning as follows.



(a) Inference time. (b) Spearman's correlation. (c) Mean and median Q-error.

Figure 6.1: Performance comparison of different tree models of the cost estimation task in TPC-DS workload.

Observation: Tree-LSTM has the best performance. Compared with other models, Tree-LSTM performs best in all evaluation indicators. The highest Spearman's Correlation and the lowest average Q-Error indicate that it can accurately capture and express critical information in most cases, even when processing complex query plans. In terms of Q-Error distribution, it has a significantly smaller tail-end error than other tree models, indicating that it is more robust in dealing with outliers and extreme situations. Experimental results demonstrate the effectiveness of its hierarchical structure-aware processing in accurately predicting complex query plan costs. However, this comes at the cost of having the highest inference overheads, as shown in Fig. 6.1(a).

Observation: GRU has a better performance than LSTM. GRU and LSTM have similar principles to capture long-term dependencies, and both demonstrate relatively good and stable performance in the experiment. However, we observe that GRU, which has a simpler architecture and fewer model parameters, has better performance than LSTM across all performance metrics. Due to its reduced overhead and enhanced performance, GRU may provide advantages in query-plan representation tasks. Its streamlined design demands less training data and time than LSTM to effectively learn dependencies between query plan nodes and can achieve shorter inference times and higher prediction accuracy.

Observation: Self-attention enhances LSTM. Comparing the results of LSTM and LSTM+Self-Attention in the experiment, we observe that integrating LSTM and self-attention as a tree model has better cost estimation performance than the basic LSTM. This means that using the Self-attention mechanism to aggregate hidden states of all nodes can more effectively aggregate node features and better represent the query plan than relying solely on the final node's state. Such findings show the potential of self-attention mechanisms in improving the tree model, demonstrating their capability to enrich model understanding and performance in complex query plan representation tasks. This suggests that employing GRU instead of LSTM for query optimization cost estimation might be a more effective choice.

Observation: TCNN seems to have relatively inferior performance. Although TCNN is currently the most advanced and widely used tree model with a unique mechanism

Table 6.2: Plan Suboptimality performance of tree models in TPC-DS workloads. The best-performing model for each metric is highlighted in **bold and underlined**.

Tree Model	Median Plan Subopt	90th Plan Subopt	99th Plan Subopt	Top 50% Mean Plan Subopt	Top 90% Mean Plan Subopt	Top 99% Mean Plan Subopt
GRU	1.049	2.478	97.991	1.013	1.139	1.581
LSTM	1.052	2.447	87.656	1.013	1.143	1.637
LSTM + Self-Attention	1.051	2.593	102.315	1.013	1.146	1.881
Tree-LSTM	1.047	2.551	**72.518**	1.012	1.137	1.824
TCNN	1.054	2.610	93.157	1.014	1.161	1.923
GNN + AddPool	1.052	2.543	100.953	1.013	1.148	1.700
Bidirectional GNN + GRU	**1.045**	**2.094**	85.441	**1.011**	**1.122**	**1.516**

designed explicitly for query plan trees, it exhibits a relatively large Q-error end-tail and slightly lower correlation compared with other evaluated models in this experiment, which means that TCNN may not be suitable for this way of encoding query plan features or may face challenges in expressing extensive and complex query plans.

6.2.3 GNN-based Tree Model Cost Estimation Performance and Analysis

We explore and evaluate the impact of using different edge structures and aggregation methods on the representation ability of GNN-based tree models. We present the experiment results of these models in the lower part of Table 6.1.

Observation: Using GRU as an aggregation method works better than conventional methods. Compared with using global add pooling, the experiment results indicate a significant improvement in the performance of GNN models across all evaluation metrics by employing GRU as the aggregation operator. The improvement verifies our perspective discussed in Section 4.1.2. Expanding the tree structure graph through post-order traversal enables GRU to learn dependencies between nodes in the actual order of how nodes are executed in the DBMS [83] and aggregate the node features utilizing GRU's gating mechanism. By leveraging the GRU for aggregation, the generated query plan representation can preserve more structural information and more valuable node features, thereby enhancing the model's ability to predict cost estimation accurately.

Observation: Utilizing the direction of the edges in the query plan tree improves the representation performance of GNN models. Our experiments demonstrate that simply using single-directed or undirected edges does not have a noticeable impact on the representation performance of the GNN-based tree models. However, models employing weighted directed GNN all show significant cost estimation performance improvements, which both promote message passing between nodes in the tree structure graph and allow nodes to learn the parent-child relationship through weighted directed edges, thereby enabling the model to capture more accurately structural and logical information of query plan even in complex workload representation tasks.

Observation: Bidirectional TransformerConv + GRU Aggregation Operator performs much better than other tree models. Our experimental findings confirm that the novel architecture that we introduced, combining bidirectional TransformerConv

Table 6.3: Runtime performance of tree models in robustness evaluation in TPC-DS workload. The first two columns present the proportion of queries where the execution time of the model-selected plans either improved or regressed compared to PostgreSQL's selection. The third and fourth columns show the ratios of the total execution times for the model-selected plans relative to PostgreSQL's choices and the actual optimal plans across all queries, respectively. The best-performing model for each metric is highlighted in **bold and underlined**.

Tree Model	Query Runtime Improved Ratio (Compared with PostgreSQL)	Query Runtime Regressed Ratio (Compared with PostgreSQL)	Total Runtime Ratio (Compared with PostgreSQL)	Total Runtime Ratio (Compared with Actual Optimal Plan)
PostgreSQL	-	-	1	1.608
GRU	0.502	0.386	0.929	1.494
LSTM	0.503	0.385	**0.877**	**1.412**
LSTM + Self-Attention	0.493	0.396	0.911	1.465
Tree-LSTM	0.505	0.375	0.925	1.486
TCNN	0.491	0.400	0.883	1.420
GNN + AddPool	0.504	0.374	0.907	1.458
Bidirectional GNN + GRU	**0.527**	**0.362**	0.884	1.422

and GRU as the aggregation operator, significantly outperforms other GNN-based as well as the state-of-the-art non-GNN tree models. Although it requires a higher inference overhead for running two complete GNN layers, it undoubtedly demonstrates the great potential and research value of GNN technology in query plan representation.

6.2.4 Plan Selection Performance and Analysis

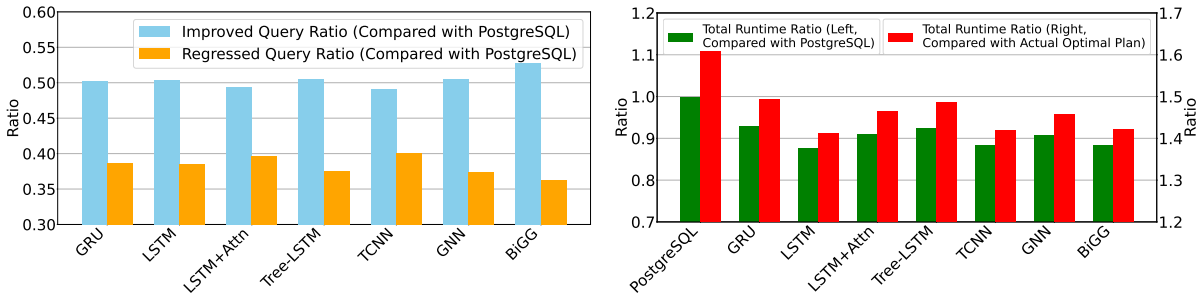
We evaluate the impact of different tree models on plan selection performance when there are multiple candidate plans. The experimental results on Plan Suboptimality and Runtime are shown in Table 6.2 and Table 6.3, respectively. The best results in each column of the two tables are underlined. In Table 6.3, the first and second columns present a comparison between the actual execution time of the optimizer's selected plan and that of PostgreSQL's plan of each query. These columns show the proportion of test queries for which the optimizer's plan either improved or regressed in performance compared to PostgreSQL's selection. The detailed ratio of these improvements and regressions is visualized in Figure 6.2(a). The third and fourth columns report the ratios of the total execution times across all queries. Specifically, the third column shows the ratio of the total execution time of the optimizer's selected plans to that of PostgreSQL's plans. The fourth column presents the ratio of the total execution time of the optimizer's plans to the actual optimal plans. These ratios help assess the cumulative impact of the optimizer's decisions over the entire workload, with the results visualized in Figure 6.2(b).

Observation: The impact of tree models on the plan selection of the optimizer is not significant. Experimental data shows that the impact of different tree models on the plan selection task of the optimizer is subtle. Although the models employ various strategies to learn query plan representations, the choice between pure tree models does not result in a sharp contrast in the optimizer's ability to select the plan closest to the

optimal plan or significantly reduce the overall run time. This subtlety in impact suggests that factors outside the tree model, such as the design of other components within the optimizer or the specific optimizations based on the workload's characteristics, may also play a crucial role in the overall robustness and effectiveness of the optimizer.

Observation: Better plan suboptimality does not lead to better total runtime. The bidirectional GNN tree model we proposed achieved the best results in all evaluation metrics of Plan Suboptimality, but this was not reflected in the improvement in the total running time. In contrast, the LSTM model, which shows less dominance in Plan Suboptimality, emerges as the most effective in reducing the total running time. This reveals an insight: while the tree model based on bidirectional GNN improves the model's ability to select a plan closest to the optimal plan, it is sometimes biased to select a more time-costly plan. This phenomenon largely stems from the characteristics of plan suboptimality. The closer the suboptimality is to 1, the more capable the model is of selecting a plan that approaches the optimal plan. However, this introduces a problem. For query plans with significant variations in execution time costs, the actual execution time differences represented by the similar plan suboptimality can vary greatly. For instance, consider a costly query where the model-selected candidate plan execution time is 15,000 ms, while the actual optimal plan executes in 10,000 ms. Here, the plan suboptimality is 1.5, resulting in a 5,000 ms time difference. In contrast, another less costly query plan might have an optimal execution time of only 100 units, and a suboptimality of 1.5 would mean just a 50 ms difference. Even though both scenarios share the same suboptimality value, the actual impact on total running time is significantly different. Moreover, consider two models, both with an average plan suboptimality of 1.5 across 2 queries: the first model selects a plan with an actual execution time of 12,000 ms for a query whose optimal execution is 10,000 ms, achieving a suboptimality of 1.2. The second model chooses a plan executing in 180 ms for another query with an optimal time of 100 ms, achieving a suboptimality of 1.8. The total running time for the first scenario is 12,180 ms, whereas in the second, if the costlier query's selected plan has a suboptimality of 1.8, resulting in 18,000 ms, and the cheaper query's selected plan has a suboptimality of 1.2, resulting in 120 ms, the total is 18,120 ms. This shows that although both models have the same average plan suboptimality, the total running time for the plans selected by the second model is nearly 1.5 times that of the first. This indicates that for queries with high execution costs, the impact of plan selection on total running time is disproportionately large compared to its impact on average plan suboptimality. If a model's training criterion is to minimize plan suboptimality, it might successfully identify the optimal plan more frequently but still perform unsatisfactorily in terms of overall running time. Therefore, in practical applications, the goal of an optimizer model is not just to approximate the optimal plan but to minimize the actual cost of query execution. Balancing both aspects will be vital in enhancing the plan selection task performance of query optimization strategies.

Observation: Cost estimation performance improvements do not imply plan selection improvements. In the experimental results shown in Table 6.1, the tree model we proposed based on bidirectional GNN has a cost estimation performance significantly ahead of other tree models. Although this trend is still reflected in the plan selection evaluation matrix, it can no longer widen the gap with the performance of other models. This



(a) Query runtime changed ratio.

(b) Total runtime ratio.

Figure 6.2: Runtime performance comparison of different tree models in TPC-DS workloads.

indicates that merely enhancing the model's cost estimation accuracy may not actually improve the optimizer's real-world performance. This indicates that the enhancement in cost estimation accuracy provided by the tree model is insufficient to substantially influence the cost model's decisions during plan selection. Furthermore, it suggests that merely relying on cost estimation may not markedly improve plan suboptimality performance when the model selects plans. Additional methods may be necessary to aid the model in making more precise decisions. This also validates our motivation to incorporate uncertainty assessment to enhance the model's performance in plan selection.

6.2.5 Conclusion

In this section, We conducted a comparison and analysis of the performance and mechanisms of the current tree models used in query plan representation in complex workloads under tasks of cost estimation and plan selection. Subsequently, We explore the possibility of applying GNN technology to query representations and introduce an innovative tree model leveraging bidirectional GNN to learn node representations and a GRU as an aggregation operator. Through extensive experiments, our proposed model was confirmed to markedly enhance the query plan representation and cost estimation accuracy compared to existing models. Therefore, in the following experiments, we will use this tree model based on bidirectional GNN and GRU as the query representation module and explore applying it to a complete learning-based query optimizer to further advance its performance in plan selection and robustness.

6.3 Proposed Model Evaluation

In the previous section, we evaluated and compared the performance of existing and our proposed tree models for query optimization, focusing on cost estimation and plan selection. During this analysis, we observed that while our tree model, which utilizes a bidirectional GNN and GRU, significantly enhances the cost model's performance in cost estimation,

the gains in plan selection are slightly improved, as discussed in Section 6.2.3 and Section 6.2.4. In this section, given the known impact of the tree model, we further evaluate the cost estimation and robustness performance improvement of our innovative cost model Reqo's architecture, which is designed to incorporate ranking-to-learn uncertainty-aware strategies. Additionally, we examine the impact of our proposed explainability technique on the model's explainability and cost estimation accuracy.

6.3.1 Experimental Methodology

Comparison. In the experimental study, we compare our proposed cost model Reqo to a classic and commercial RDBMS optimizer PostgreSQL, as well as three prominent recent works: Bao [88], Lero [182], and Roq [58]. Bao was chosen for its advanced performance and demonstrated robustness to errors in input features. Lero was selected due to its use of a learning-to-rank mechanism. Roq was included by its approach to quantifying uncertainty for robust plan selection. These comparisons allow us to evaluate how our model improves across different aspects based on these state-of-the-art models using different mechanisms. In this process, we also segment our model Reqo to evaluate how each component of our proposed architecture influences the model's overall performance. All query plans used in this study are executed within the PostgreSQL execution engine to obtain the actual workloads. In experiments for robustness evaluation, each approach is evaluated for its ability to select a robust plan from the same set of plans for a given query. The search space for each query and corresponding datasets are detailed in previous sections. In our experimental study focusing on the explainability of the query optimization cost model, we also compare the explainability performance of the robust cost model integrated with our proposed explainability technique and that of other state-of-the-art approaches during the query plan cost prediction process. All approaches employed in this study utilize the same query plan datasets and the subtree extraction strategy mentioned in Section 4.2.1 while comparing. All query plans used as samples are executed using the PostgreSQL execution engine to obtain workloads, and the total actual execution times for each node and all its child nodes, are recorded as explanation labels. Furthermore, the explainability technique we propose is integrated into our robust cost model to assess the effects of this integration on model performance and explainability. The details of approaches used in the experimental comparison are shown as follows:

PostgreSQL is an advanced, enterprise-class open-source relational DBMS, which represents the state-of-the-art classical and commercial query optimizer. Its performance in cost estimation and plan selection is used as a baseline in our experiments. We get PostgreSQL's estimated execution cost of each plan in the dataset by the command 'Explain'. By comparing our model's performance against PostgreSQL's, we can ensure that our proposed model provides not only more accurate cost estimation but also more robustness. Additionally, PostgreSQL's query optimizer is less influenced by machine learning biases, making it a strong, traditional benchmark to evaluate the performance of learned models in real-world database systems. Besides, it is also used as the baseline in our explainability experiments. Through the 'Explain' command, the PostgreSQL optimizer can provide the total estimated cost of each node and all its child nodes in the plan, which can be

utilized to calculate the contribution of subgraphs to explain the predictions and evaluate the performance of PostgreSQL itself in cost estimation and explainability. These results from PostgreSQL are used as the baseline. The comparison allows us to explore the explainability gap between traditional query optimizers and learning-based approaches, and to quantify the improvements achieved by our proposed techniques.

Bao is a learned query optimizer that enhances query optimizers by applying query-specific hints using reinforcement learning techniques. Here, we use its cost model part, which employs a learned approach that processes the vectorized plan tree through TCNN layers to predict latency. Each node feature in the vectorized plan tree contains information about the node type, along with PostgreSQL's estimated cardinality and cost.

Lero is a learning-to-rank query optimizer. Similar to Bao in its plan encoding approach, Lero diverges from typical learned cost models that aim to predict latency values. Instead, it trains its cost model to identify the more effective plan in a pair, framing this as a binary classification task. This method is generally simpler and more efficient than the regression tasks of predicting cost or latency in terms of model performance and accuracy. However, due to this characteristic, it was excluded from the comparison of cost value prediction performance in our experiments.

Roq is a framework designed for robust query optimization using a risk-aware learning approach. It employs a GNN model to create query-level encodings based on the join graph and a plan encoding method similar to Bao. These encodings are then input into the cost model to estimate latency and quantify the associated risks of data and model uncertainties. Roq incorporates strategies to use these quantifications for robust plan evaluation and selection.

Our proposed cost model (Reqo) is segmented into various parts according to its architecture, with each subsequent part incorporating an additional component or mechanism compared to its predecessor. This incremental evaluation approach allows us to effectively verify the influence of each component on the overall framework. The specific configurations of the models used in this experiment are shown as follows:

1. **Base model (LogMSE Loss).** The basic model here refers to a cost model consisting of the learning-based Feature Encoder described in Section 5.1, our proposed tree model based on bidirectional GNN and GRU as discussed in Section 4.1, and a straightforward estimator only based on MLP. The loss function used is the LogMSE loss function defined in Equation 6.10 based on the mean square error of estimated and actual cost. The entire architecture and loss function of this base model does not involve the quantification and use of uncertainty, which is the same model architecture used in the tree model comparison mentioned in Section 6.2. The purpose of using the base model is to verify whether the proposed mechanisms we added in the following parts have truly improved the performance of the cost model.

2. **Proposed model (Uncertainty Loss).** The cost model used in this segment utilizes the same feature encoder and tree model as the above-mentioned base model, but it incorporates an estimator with two branches as described in Section 4.7, where

the first branch is dedicated to predicting the plan's expected execution time, and the second branch quantifies the uncertainty associated with the execution time. The loss function applied in this segment is the uncertainty loss function detailed in Equation 4.14. Although this segment generates a prediction of uncertainty, it does not yet apply this quantified uncertainty in the processes of cost estimation or plan selection. This setup is specifically designed to evaluate the impact of integrating an uncertainty quantification mechanism and corresponding loss function on the overall performance of the model.

3. **Proposed model (Uncertainty Loss + Fixed Parameter f_{fixed}).** In this segment, the model architecture remains identical to the previous one, employing the same feature encoder and tree model. However, after obtaining the expected execution time and uncertainty from the two-branch outputs of the estimator, we integrate the predicted execution time and uncertainty using a fixed hyperparameter f_{fixed}, which is determined through parameter tuning but not learning. This integrated value is then utilized for plan selection and robustness performance evaluation. In this segment, we do not employ the learning-based parameter $f_{learned}$ and ranking loss as described in the relevant Section 4.7. The purpose of this setup is to provide a baseline to evaluate whether our more complex, ranking-based uncertainty learning mechanism offers a significant improvement over simpler methods of incorporating uncertainty by a fixed value into plan selection decision-making processes.

4. **Proposed model (Uncertainty Loss + Learnable Parameter $f_{learned}$ + Ranking Loss) or Reqo (without explainability technique).** The model architecture here is exactly the same as the architecture of the robust cost model based on ranking-to-learn described in Section 4.7. This configuration utilizes the previously mentioned feature encoder, tree model, and two-branch cost estimator. Additionally, a learning-based parameter $f_{learned}$ is incorporated to integrate the estimated expected execution time and uncertainty adaptively. By employing a ranking mechanism, the model leverages the integrated values to learn from comparisons between different plans. This architecture aims to achieve theoretically optimal robust performance in this study by optimizing the model's plan comparison decision-making in response to the dynamic integration of execution time predictions and associated uncertainties. During this process, we do not integrate the explainability technique we proposed to the model architecture and loss function. Instead, the explanation process is implemented by directly inputting the subtrees, which are extracted from the query plan based on the strategies outlined in Section 4.2.1, into the cost model as an individual sample. This approach then outputs the execution time prediction for each subplan. At this stage, the loss function employed by the model is the uncertainty and ranking loss function used in our cost model without the integration of the explainability technique, as detailed in Equation 4.17. By analyzing the inclusion relationships among various subtrees, we can determine the contributions of subgraphs in different sizes to the final predicted value, thereby enabling the explanation of cost prediction. The objective of this segment is also to assess the explanation performance of this cost model under our evaluation criteria without the assistance

Table 6.4: Cost estimation performance results across workloads. The best-performing model for each metric within each dataset is highlighted in **bold and underlined**.

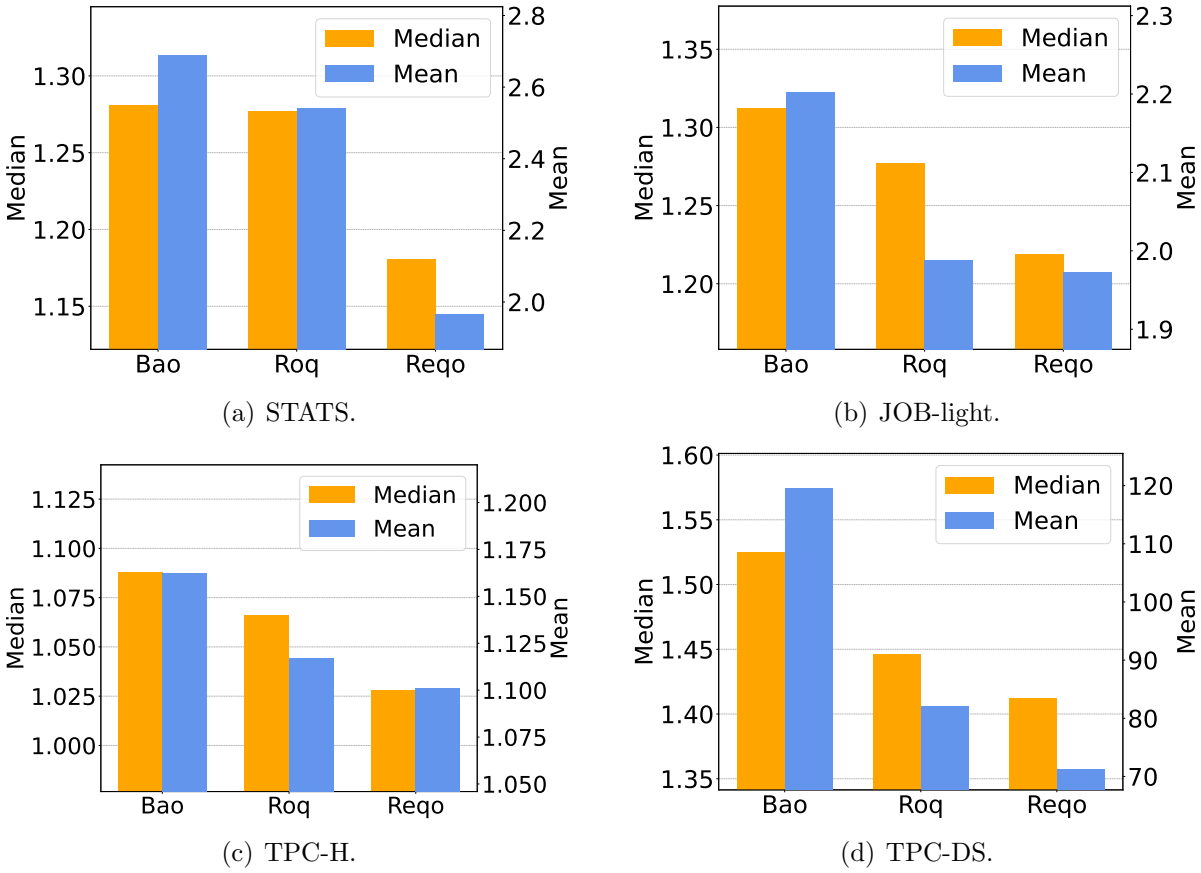
Dataset	Model	50th	90th	99th	Top50% Mean	Top90% Mean	Top99% Mean	Mean	Spearman
STATS	Bao	1.281	2.696	23.972	1.127	1.360	1.776	2.690	0.931
	Roq	1.277	2.523	21.652	1.119	1.307	1.760	2.540	0.936
	Reqo	**1.181**	**1.868**	**10.271**	**1.081**	**1.211**	**1.394**	**1.966**	**0.966**
JOB-light	Bao	1.312	2.587	19.317	1.144	1.374	1.726	2.202	0.949
	Roq	1.277	2.766	15.808	1.124	1.334	1.676	1.988	0.952
	Reqo	**1.219**	**2.404**	**14.446**	**1.114**	**1.302**	**1.624**	**1.973**	**0.954**
TPC-H	Bao	1.088	1.273	2.831	1.039	1.090	1.115	1.162	0.980
	Roq	1.066	1.201	3.628	1.031	1.069	1.089	1.117	0.978
	Reqo	**1.028**	**1.150**	**1.623**	**1.013**	**1.033**	**1.065**	**1.101**	**0.993**
TPC-DS	Bao	1.525	5.877	111.496	1.222	1.793	3.430	119.556	0.868
	Roq	1.446	5.373	119.456	1.190	1.684	3.299	82.022	0.879
	Reqo	**1.412**	**5.025**	**80.005**	**1.187**	**1.661**	**2.944**	**71.251**	**0.906**

of the explainability technique. This exploration provides a foundation to determine whether our explainability technique can really enhance the interpretability and credibility of the model.

5. **Reqo (with explainability technique).** The model we implement here integrates our proposed explainability technique with the previously described model as the entire architecture of Reqo. At this stage, the model processes a query plan and all its subtrees as input, where the plan itself and each subtree are converted into plan-level embeddings by the tree model. These subtrees' embeddings are then concatenated with the embeddings of the complete query plan to which they belong. The concatenated embeddings are input into the explainer using the explainability technique to generate the ratio of each subtree's contribution to the final predicted execution time. Through the inclusion relationships of subtrees of different sizes, we can deduce the specific total contribution value of the required subgraph for the interpretation of cost predictions for query plans. During the model training process, the contribution values estimated by the explainer are utilized through the explanation loss function. The specific loss function used combines the loss function of the cost model itself with the explanation loss function, as depicted in Equation 5.5. This approach enables the model to automatically learn how to accurately interpret the contribution proportions of different subtrees during training, while also enhancing the explainability of the plan embeddings generated by the model. Therefore, this integration theoretically improves the overall explainability and credibility of the model. In this chapter, 'Reqo' alone implies that it contains all modules, including the explainability technique.

6.3.2 Cost Estimation Performance and Analysis

Table 6.4 and Figure 6.3 present the performance of advanced cost estimation models including our proposed Reqo across various workloads. Compared to Bao and Roq, Reqo consistently outperforms these models on all cost estimation evaluation metrics. While all models perform well in simpler workloads with relatively small performance differences,



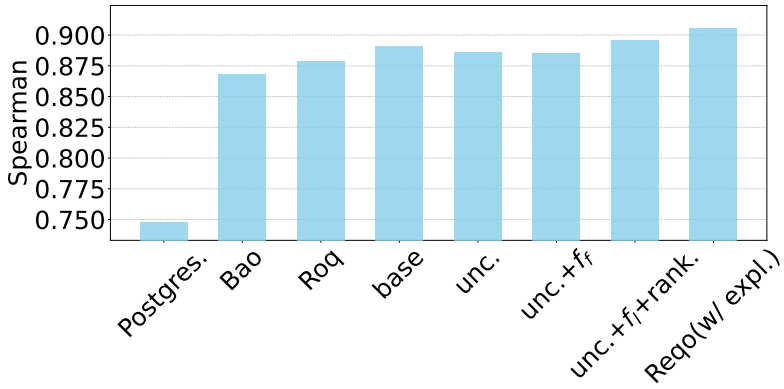
(a) STATS. (b) JOB-light.

(c) TPC-H. (d) TPC-DS.

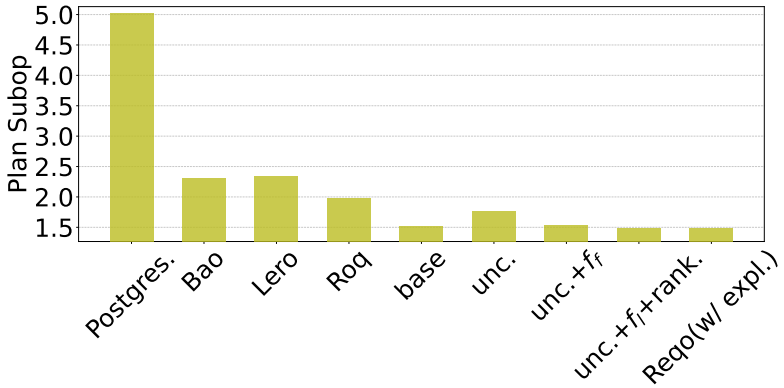
Figure 6.3: Cost estimation performance across different benchmarks.

in the more complex TPC-DS workload, Reqo distinctly excels, delivering more precise latency predictions with smaller deviations. This underscores Reqo's effectiveness and stability in complex scenarios.

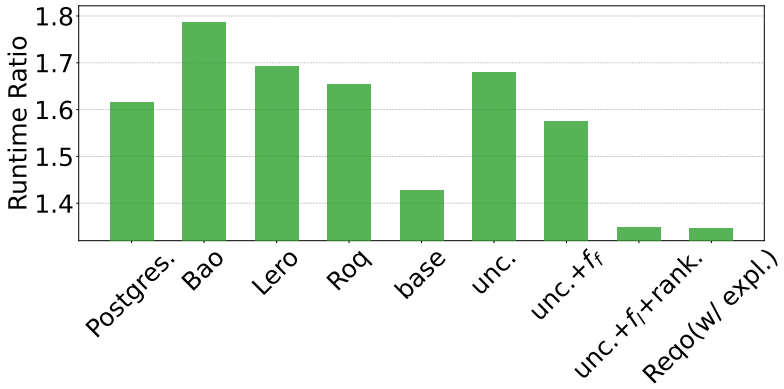
To assess the impact of our proposed mechanisms on model performance, Figure 6.4(a) demonstrates the performance variations of Reqo with different configurations and other cost models under the TPC-DS workload with a focus on Spearman's Correlation. First, we observe that all learning-based cost models significantly outperform PostgreSQL's traditional cost model. Utilizing the BiGG tree model equipped with bidirectional GNN and GRU, our base model surpasses Bao and Roq without additional mechanisms, demonstrating its powerful representational capabilities. This confirms that BiGG can capture information from the query plan more effectively and predict latency more accurately. Integrating uncertainty loss enables the model to quantify uncertainty during cost estimation for enhanced robustness in plan selection, which comes at the cost of slightly reduced estimation accuracy. However, incorporating the ranking-to-rank mechanism compensates for this drawback and further enhances the performance. This confirms that while designed to improve robustness, our learning-to-rank robust cost model still maintains strong cost estimation performance.



(a) Spearman's correlation.



(b) Plan Suboptimality (Top 99% Mean).



(c) Total Runtime Ratio (Optimal).

Figure 6.4: The impact of the different configurations of Reqo on (a) Spearman's correlation, (b) Plan Suboptimality (Top 99% Mean) and (c) the total runtime ratio (compared with the total runtime of actual optimal plans) in the TPC-DS workload.

Table 6.5: Plan suboptimality performance results across workloads. The best-performing model for each metric within each workload is highlighted in **bold and underlined**.

Dataset	Model	50th	90th	99th	Top50% Mean	Top90% Mean	Top99% Mean	Mean
STATS	PostgreSQL	1.027	1.765	5.713	1.008	1.093	1.248	1.455
	Bao	1.028	1.488	3.335	1.008	1.074	1.139	1.183
	Lero	1.023	1.423	2.805	1.006	1.055	1.116	1.151
	Roq	1.022	1.432	2.915	1.006	1.054	1.117	1.148
	Reqo	**1.019**	**1.263**	**2.597**	**1.005**	**1.035**	**1.080**	**1.115**
JOB-light	PostgreSQL	1.806	20.500	87.368	1.252	3.305	6.543	7.820
	Bao	1.231	2.949	33.233	1.076	1.323	1.921	2.340
	Lero	1.198	2.597	30.193	1.063	1.271	1.783	2.181
	Roq	1.106	1.874	**15.732**	1.025	1.162	1.377	1.632
	Reqo	**1.103**	**1.828**	18.845	**1.024**	**1.146**	**1.360**	**1.601**
TPC-H	PostgreSQL	1.032	1.317	2.592	1.008	1.059	1.134	1.150
	Bao	1.015	1.113	1.359	1.003	1.023	1.035	1.038
	Lero	1.016	1.126	1.356	1.004	1.031	1.039	1.048
	Roq	1.010	1.107	**1.241**	1.002	1.017	1.029	1.031
	Reqo	**1.008**	**1.088**	1.300	**1.001**	**1.014**	**1.027**	**1.029**
TPC-DS	PostgreSQL	1.129	6.910	303.818	1.027	1.567	5.030	150.171
	Bao	1.070	3.224	102.861	1.017	1.219	2.305	230.961
	Lero	1.061	3.081	130.764	1.015	1.197	2.351	270.757
	Roq	1.052	2.521	78.784	1.013	1.154	1.989	61.843
	Reqo	**1.043**	**1.984**	**54.610**	**1.011**	**1.112**	**1.490**	**44.645**

6.3.3 Robustness Performance and Analysis

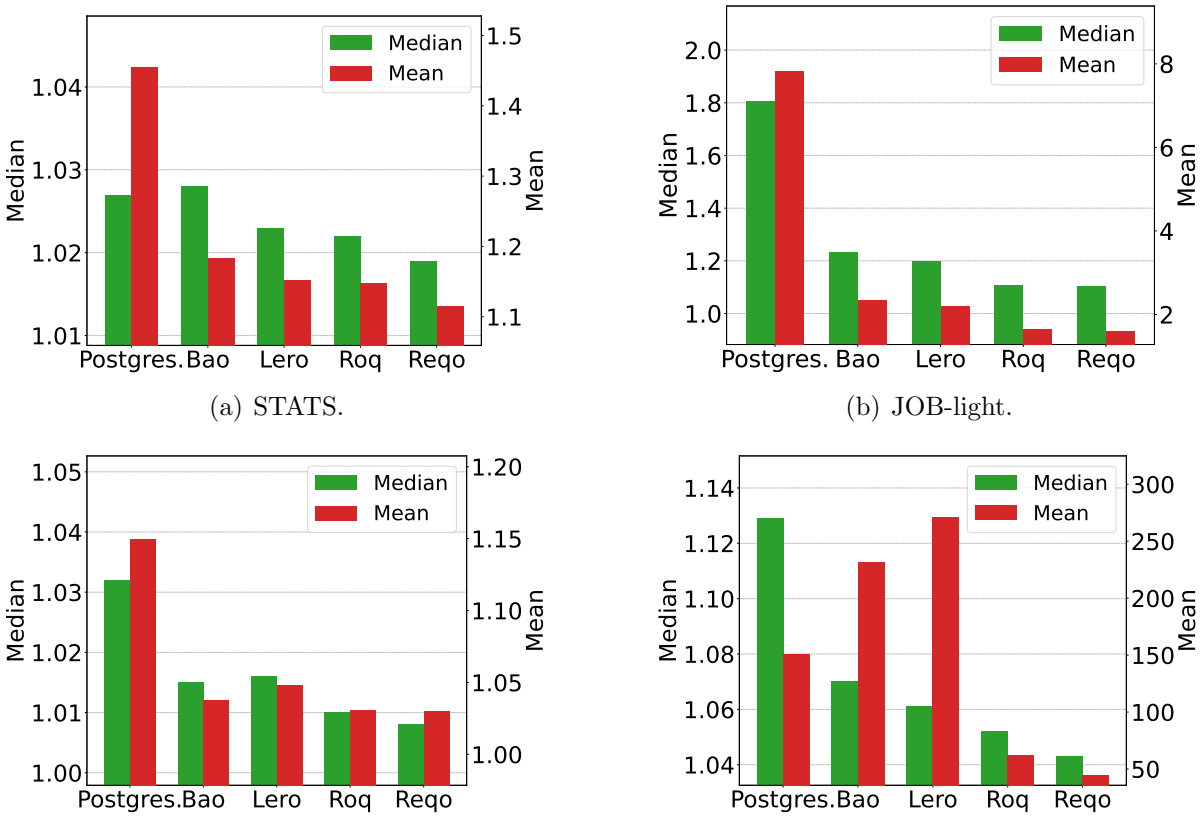
In the robustness experiments, we evaluate our proposed model and PostgreSQL in terms of plan suboptimality and runtime. The experimental results on plan suboptimality are presented in Table 6.5 and illustrated in Figure 6.5, while the experimental results on runtime are presented in Table 6.6 and Figure 6.6. We evaluated different configurations of our proposed model, Reqo, on Spearman's correlation, plan suboptimality, and total runtime, which are presented in Figure 6.4.

PostgreSQL vs Base model. Experimental results show that compared to PostgreSQL using classic optimizers, our learning-based base model has shown an overall better performance than PostgreSQL in all evaluation metrics. In terms of cost estimation, although there is no additional mechanism, benefiting from our bidirectional GNN and GRU-based tree model's advanced plan representation capabilities, the model achieves a significant 15% lead in Spearman correlation. This superior cost estimation accuracy brings improved plan suboptimality performance to varying degrees, proved by improvements across the metrics in Figure 6.4(b), particularly at the plan suboptimality distribution tails, which represents a more robust performance of the base model compared to PostgreSQL under extreme conditions. Additionally, runtime results shown in Figure 6.4(c) reveal that our base model selects a better plan than PostgreSQL in over half of the samples. The total runtime for plans selected by our base model is 88% of that chosen by PostgreSQL and is closer to the total runtime of the actual optimal plans. This comparison highlights that our base model improves robustness without slowing down the overall execution time of workloads, demonstrating the base model's effectiveness and its superiority over traditional commercial optimizers.

Base model vs Proposed model (Uncertainty Loss). The biggest difference

between the two models is the loss function they use: the base model employs a LogMSE loss function (Equation 6.10), while the proposed model utilizes an uncertainty loss function (Equation 4.14) based on a Gaussian prior. Notably, the proposed model does not use the output of its second branch, which is dedicated to uncertainty prediction. Thus, the comparison of these two models here aims to explore the impact of the loss function itself on the model performance. The experimental results indicate that using the uncertainty loss function's execution time estimation output alone without utilizing its uncertainty results in a poorer overall performance compared to the base model, which is reflected in the two main metrics. The use of uncertainty loss leads to a slight degradation in all metrics of plan suboptimality relative to the base model. Moreover, in runtime performance, the proposed model underperforms not only relative to the base model but also slightly worse than PostgreSQL, particularly in the total runtime of the selected plans. However, cost estimation performance does not show significant degradation, suggesting that the proposed model effectively learns from the plan embeddings to achieve relatively accurate cost predictions by uncertainty loss. Despite this, it shows less robustness in extreme cases and is less able to reduce the total runtime of the workload. This illustrates a limitation of using the uncertainty loss function for learning: while it enables the model to learn how to quantify uncertainty, it comes at the expense of accuracy and robustness in execution time prediction of the model's first branch to some extent. Therefore, how to utilize the uncertainty in its output through certain mechanisms to overcome its own limitations and even further improve the model performance is the main issue that needs to be addressed for the application of uncertainty quantification in the cost model.

Proposed model (Uncertainty Loss) vs Proposed model (Uncertainty Loss + Fixed Parameter f_{fixed}). The distinction between these two models is in their utilization of the predicted uncertainty: the first model completely ignores the uncertainty from the second branch, while the second model attempts to integrate the expected execution times and corresponding uncertainties through a simple method, weighting them with a fixed parameter f_{fixed}. This comparison is designed to explore whether utilizing uncertainty for plan selection can improve the model's robustness in plan selection. In terms of plan suboptimality experimental results, the model using the fixed parameter f_{fixed} for integrating outperforms the model that ignores uncertainty but shows similar performance to the base models, which do not incorporate an uncertainty quantification mechanism at all. In the runtime experimental results, the model using f_{fixed} also performs slightly better than the model without using f_{fixed}. However, it still performs worse than the base model, particularly in terms of total runtime ratio. The performance of Spearman correlation in cost estimation is not affected since the uncertainty and the fixed parameter f_{fixed} are not involved in the model's training phase, which remains slightly inferior to the base model. These findings highlight a phenomenon that integrating uncertainty and estimated execution time using a fixed parameter f_{fixed} does improve robustness in plan selection compared to models ignoring uncertainty. However, it still fails to meet the performance standards set by the base model. This suggests that while using quantified uncertainty does enhance model robustness, using a simplistic fixed-value integration obtained through hyperparameter tune fails to fully exploit the potential of uncertainty, resulting in only marginal improvements in model performance. It brings a new challenge:



(a) STATS.

(b) JOB-light.

(c) TPC-H.

(d) TPC-DS.

Figure 6.5: Plan suboptimality performance of various models across different workloads.

a more complex and adaptive method is needed to utilize uncertainty effectively in order to maximize the benefits of plan selection and overall model robustness.

Proposed model (Uncertainty Loss + Fixed Parameter f_{fixed}) vs Proposed model (Uncertainty Loss + Learnable Parameter $f_{learned}$ + Ranking Loss). At this stage, we deploy the complete architecture of our robust cost model based on ranking-to-learn, which as opposed to using a fixed parameter f_{fixed} to integrate the estimated execution time and its uncertainty, introduces a learning-based mechanism that replaces f_{fixed} with a learnable parameter $f_{learned}$. This parameter $f_{learned}$ and the predicted uncertainty are integrated into the model's training process through the ranking loss function, enabling adaptive learning. This mechanism allows the model to automatically learn how to refine its uncertainty predictions more effectively, and the plan comparisons based on integrated values enable the model to fully exploit the obtained uncertainty, thereby improving the robust performance of the model. Our experimental results corroborate this theory. Among all the evaluation metrics of plan suboptimality and running time, our proposed cost model with the ranking mechanism achieves the best performance compared with all the approaches, and its improvement in the total running time is especially obvious, which definitely proves the architecture's advanced capability in improving the robustness

Table 6.6: Runtime performance results across workloads. The best-performing model for each metric is highlighted in **bold and underlined**.

Dataset	Model	Query Runtime Improved Ratio	Query Runtime Regressed Ratio	Total Runtime Ratio (PostgreSQL)	Total Runtime Ratio (Optimal)
STATS	PostgreSQL	-	-	1.0	1.603
	Bao	0.302	**0.299**	0.804	1.289
	Lero	0.323	0.331	0.801	1.284
	Roq	0.470	0.392	0.800	1.282
	Reqo	**0.535**	0.338	**0.752**	**1.205**
JOB-light	PostgreSQL	-	-	1.0	1.938
	Bao	0.524	0.134	0.692	1.341
	Lero	0.557	**0.122**	0.655	1.269
	Roq	0.744	0.144	0.629	1.219
	Reqo	**0.764**	0.149	**0.628**	**1.217**
TPC-H	PostgreSQL	-	-	1.0	1.139
	Bao	0.228	0.320	0.903	1.028
	Lero	0.200	**0.173**	0.925	1.054
	Roq	**0.497**	0.281	0.898	1.023
	Reqo	0.487	0.207	**0.894**	**1.018**
TPC-DS	PostgreSQL	-	-	1.0	1.615
	Bao	0.434	0.346	1.106	1.786
	Lero	0.450	0.338	1.048	1.693
	Roq	0.513	0.376	1.025	1.655
	Reqo	**0.541**	**0.338**	**0.834**	**1.347**

of the cost model. As for the results of the cost estimation performance experiments, the model with the full architecture achieves a slightly higher Spearman correlation than the base model despite using an uncertainty loss function. This further illustrates the adaptive $f_{learned}$ and uncertainty in the ranking mechanism, as well as the model's ability to discriminate between different plans directly learned from plan comparisons, enabling the model to overcome the loss of performance brought by quantitative uncertainty. Consequently, our proposed ranking-to-learn cost model excels in cost estimation and demonstrates remarkable robustness, showcasing the synergistic impact of adaptive integration of uncertainty and plan comparison in enhancing the overall model performance.

Plan Suboptimality Performance. We evaluate the robustness performance of our proposed Reqo model alongside other advanced cost models using plan suboptimality. The experimental results on plan suboptimality are presented in Table 6.5 and Figure 6.5. The experimental results on plan suboptimality demonstrate that the learning-based approaches generally outperform the classical optimizer utilized by PostgreSQL. In simpler workloads, it is notable that the performance difference between the mean from the top 50%, even 90% and the median of their plan suboptimality is actually very small. This suggests that these approaches are already able to select the optimal or near-optimal plans in most of the samples, and thus it becomes more difficult to improve them further. The mean plan suboptimality, however, serves as an indicator of their performance under extreme conditions or with particularly complex queries, and thus more indicatively reflects the robustness of the models. The fact that our proposed Reqo shows improvements across all metrics confirms its efficacy in enhancing cost model robustness performance and highlights that Reqo achieves the most accurate plan selection with the minor tail. This architecture not only handles typical scenarios effectively but also demonstrates its strength in more

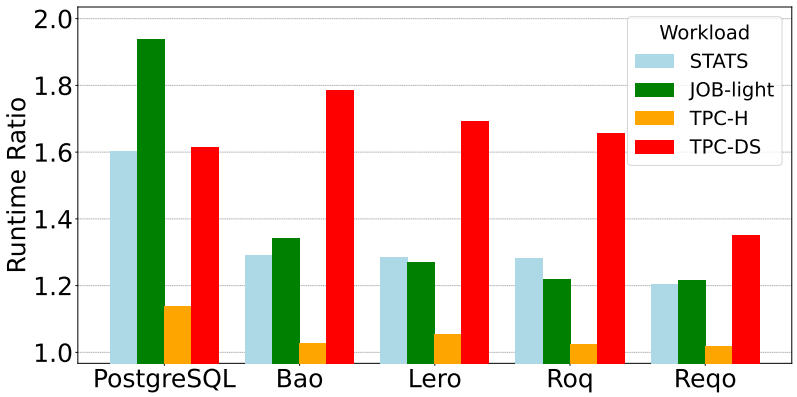


Figure 6.6: Comparing the total runtime ratio performance (ratio of the total runtime of the model selected plans to the total runtime of actual optimal plans) of cost models across different workloads.

challenging conditions, highlighting its comprehensive capability to improve plan selection and optimize query processing performance. This also verifies our previous point that modules other than the tree model are also important for improving the robustness of the cost model. In the more complex TPC-DS workloads, Reqo and Roq, which employ uncertainty quantification mechanisms aimed at enhancing the robustness of plan selection, significantly outperform other models. Reqo achieves the best performance that exceeds Roq by utilizing a more advanced ranking-based adaptive integration of uncertainty and latency, thereby confirming its superior robustness in plan selection.

Runtime Performance. The experimental results in runtime appear in Table 6.6 and Figure 6.6. Compared to plan suboptimality, runtime more directly reflects the model's optimization in terms of time cost across the entire workload. In the runtime experimental results, Reqo consistently outperforms other models across nearly all evaluation metrics, demonstrating substantial performance enhancements. Notably, in the more complex TPC-DS workload, models like Bao, Lero, and Roq do not perform as well as PostgreSQL in terms of total runtime but show significantly better performance in simpler workloads. This suggests that these models struggle with more complex query plan trees. However, our advanced feature encoder and BiGG's robust representation capabilities, coupled with the ranking-to-rank uncertainty quantification mechanism, ensure that Reqo maintains superior performance even with complex query plans. Specifically, in optimizing total runtime, Reqo achieves a performance enhancement of 16.6% over PostgreSQL, 24.6% over Bao, 20.4% over Lero, and 18.6% over Roq. These results underscore Reqo's robustness and efficiency in handling complex query scenarios, firmly establishing its superiority in runtime performance optimization.

Although our proposed architecture has a significant improvement in terms of total runtime ratio, it has only a weak leading trend query runtime changed ratio. Despite our proposed architecture being able to make better selections than PostgreSQL in most cases, the existence of a regressed runtime ratio demonstrates that traditional cost models have always outperformed learning-based approaches when dealing with certain kinds of query plans. The identification of such query plans and their targeted processing is one of the

Table 6.7: Explanation performance results across workloads. Top1 Acc, Top1and2 Acc, and Top1or2 Acc Ratio evaluate the model's explanation accuracy in identifying the most influential subgraphs, with Top1 Acc focusing on the single most significant subgraph, Top1and2 Acc on the top two, and Top1or2 Acc on either of the top two. Top1 Infl., Top1and2 Infl. measure the explanation subgraph influence ratio of these model-selected most influential subgraph(s) total contribution to the actual most influential subgraph(s). The best results in each category are **bold and underlined**.

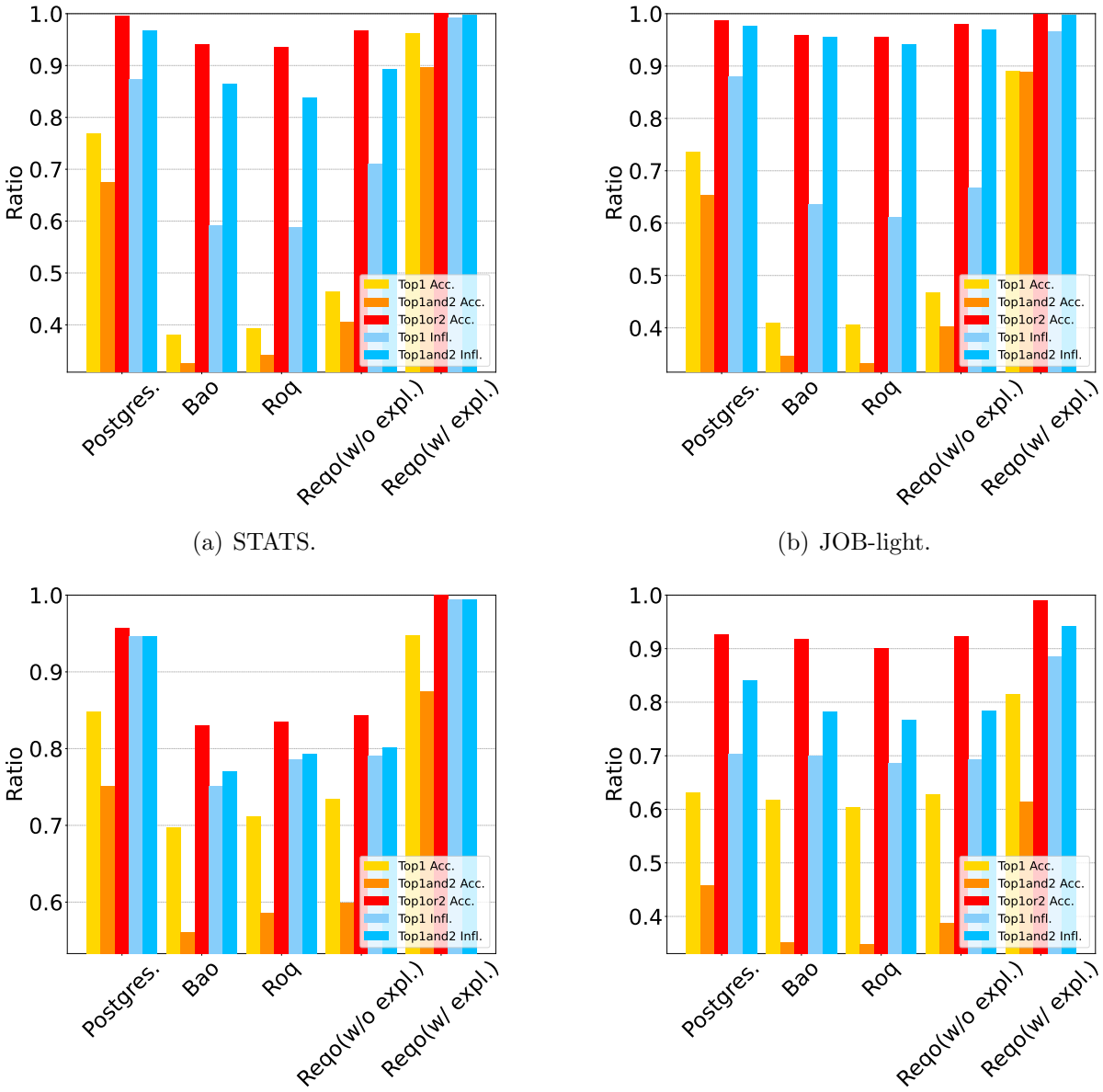
Dataset	Model	Expl Top1 Acc Ratio	Expl Top1and2 Acc Ratio	Expl Top1or2 Acc Ratio	Expl Top1 Infl Ratio	Expl Top1and2 Infl Ratio
STATS	PostgreSQL	0.769	0.675	0.996	0.873	0.967
	Bao	0.381	0.325	0.941	0.592	0.865
	Roq	0.392	0.341	0.935	0.588	0.837
	Reqo (w/o expl)	0.464	0.405	0.967	0.710	0.892
	Reqo (w/ expl)	**0.962**	**0.897**	**1.000**	**0.992**	**0.997**
JOB-light	PostgreSQL	0.737	0.654	0.988	0.881	0.976
	Bao	0.409	0.346	0.960	0.636	0.956
	Roq	0.407	0.332	0.955	0.611	0.942
	Reqo (w/o expl)	0.468	0.402	0.981	0.668	0.969
	Reqo (w/ expl)	**0.890**	**0.889**	**1.000**	**0.967**	**0.998**
TPC-H	PostgreSQL	0.848	0.751	0.957	0.947	0.946
	Bao	0.697	0.561	0.830	0.751	0.771
	Roq	0.712	0.586	0.835	0.786	0.793
	Reqo (w/o expl)	0.734	0.599	0.844	0.791	0.802
	Reqo (w/ expl)	**0.948**	**0.875**	**1.000**	**0.994**	**0.995**
TPC-DS	PostgreSQL	0.631	0.458	0.926	0.703	0.841
	Bao	0.617	0.351	0.918	0.699	0.782
	Roq	0.603	0.348	0.901	0.685	0.766
	Reqo (w/o expl)	0.628	0.387	0.922	0.693	0.784
	Reqo (w/ expl)	**0.814**	**0.613**	**0.989**	**0.885**	**0.941**

very promising solutions to improve the robustness of the cost model in the future.

6.3.4 Explainability Performance and Analysis

In this section, we evaluate PostgreSQL and our proposed cost model with our explainability technique in terms of their explainability regarding execution time prediction. The experimental results on explanation accuracy and subgraph suboptimality are presented in Table 6.7, which are visualized in Figure 6.7. The first three columns of this table are the evaluation results of the model's ability to accurately identify the most influential subgraphs—specifically in the three scenarios, the top one, the top two, and at least one of the top two most influential subgraphs, respectively. Additionally, columns 4 and 5 show the results of subgraph suboptimality, focusing on the ratio of total execution time required for the top one or top two most influential subgraphs selected by the model, along with the subgraphs that actually contribute the most.

PostgreSQL vs Learning-based Model (Without explainability technique). From the explanation experimental results as shown in Table 6.7 and Figure 6.7, we observe that the PostgreSQL traditional optimizer does not show a performance drop as seen in previous experiments but instead exhibits relatively good explainability. Leaning-based



(a) STATS. (b) JOB-light.

(c) TPC-H. (d) TPC-DS.

Figure 6.7: Comparing the explanation performance of models across different workloads.

cost model, including our proposed Reqo (without explainability technique), achieves excellent performance in both cost estimation accuracy and robustness, which are significantly higher than PostgreSQL but worse than PostgreSQL in terms of cost prediction explainability. The main reason is that these models prioritize how to transform the query plan into a high-qualified plan-level embedding to predict cost estimation rather than identifying specific subgraphs of the plan that significantly influence the embedding and drive the model to make such a prediction. This issue is a common limitation across nearly all current learning-based cost models, which provide accurate cost estimates but lack the ability to explain why certain decisions are made, hindering targeted query optimization. In

contrast, the classical optimizer used by PostgreSQL bases its cost predictions on detailed cost statistics for each node operation in the plan, enabling it to show its decision-making process clearly and perform better in this comparison. The experiment results demonstrate that Without using our proposed explainability technique, the learning-based cost models perform worse than PostgreSQL in almost all explanation evaluation metrics. In particular, there is a clear gap in accuracy when the learning-based cost model is required to find the two most influential subgraphs in sequence simultaneously. This suggests that although these cost models can provide vague explanations, it's difficult for them to pinpoint subgraphs' contributions precisely. This lack of accuracy in explanation brings challenges for trusting the results of query optimization in practice and illustrates the necessity of integrating an explainability mechanism into the query plan cost prediction process.

PostgreSQL vs Reqo (With explainability technique). Due to the limitations of the traditional optimizer itself, PostgreSQL's cost estimate for each node is not accurate, which leads to its explanation performance not being accurate enough. The integration of explainability techniques enhances the transparency and accuracy of cost predictions made by learning-based cost models, similar to traditional models such as PostgreSQL. Experimental findings demonstrate that Reqo, when equipped with explainability features, surpasses other models across all evaluative metrics. Specifically, in simpler scenarios, it achieves perfect scores in the Top 1 or 2 accuracy ratio and nearly perfect scores in the Top 1 and 2 influence ratio, effectively identifying and quantifying the most influential subgraphs. Even in the challenging TPC-DS workload, Reqo demonstrates substantial enhancements, with explanation accuracy metrics improving by nearly 20% and explanation influence ratios exceeding a 10% increase compared to PostgreSQL. Therefore, we can affirm that our explainability technique based on query plan subtrees effectively enhances the explainability of the cost model. The cost model integrated out explainability technique can adaptively learn the correlation between the embeddings of the subgraphs and their corresponding complete query plan during the training process, enabling a more precise estimation of the subgraphs' contributions to the execution time predictions and, thereby, a more accurate explanation of the prediction results. Additionally, the inclusion of explanation loss allows the integrated cost model to focus more on the process of generating the embedding itself, which means the greater the subgraph's contribution, the greater the correlation between the embedding of the subgraph and the embedding of the complete plan. This mechanism not only enhances the explainability of the embeddings themselves but also ensures that the model can more effectively understand and explain the significance of each part of the query plan, further improving overall model explainability.

Additionally, as evident from Figure 6.4, integrating explainability technology does not compromise Reqo's cost estimation and robustness performance; it even slightly enhances it. This enhancement partly depends on the explainability technology that preserves the original model architecture while improving the model's comprehension and representation of query plans. Moreover, the process of extracting subtrees for explainability not only utilizes the information within the query plan more effectively but also acts as a form of data augmentation.

6.3.5 Conclusion

The experimental results strongly validate the effectiveness of our proposed cost model architecture. Our model not only demonstrates significant improvements in cost estimation accuracy but also enhances the robustness of query plan selection, ensuring reliable performance even in complex scenarios. The integration of the learning-to-rank mechanism with uncertainty quantification emerges as a key innovation, allowing the model to autonomously balance cost estimates with associated uncertainties. This capability enhances the model's ability to select more robust execution plans by minimizing the impact of estimation errors, ultimately leading to superior query optimization performance.

Moreover, the experiments affirm the value of incorporating uncertainty quantification directly into the model's training process. This holistic approach ensures that the model learns to leverage uncertainty as an asset rather than a threshold, fostering more adaptive and effective decision-making during plan selection. This marks a significant step forward in the development of learning-based query optimizers, offering a promising solution for mitigating the inherent uncertainty challenges in query optimization.

The introduction of explainability techniques further enriches the performance of our model by enhancing transparency and interpretability. Through a novel subgraph extraction strategy tailored to query plan cost estimation, the model identifies key substructures that influence cost predictions. This is the first study to explain the predictions of learning-based query cost models using meaningful subgraphs, making the optimizer more interpretable and trustworthy for users and administrators. The ability to understand the impact of different subgraphs contributes not only to improved explainability but also positively affects the model's cost estimation accuracy and credibility.

Additionally, the combination of the three proposed mechanisms—the bidirectional GNN-based representation model, the uncertainty-aware cost model with learning-to-rank capabilities, and the explainability technique—provides a comprehensive improvement across multiple performance dimensions. These integrated innovations result in a robust, accurate, and explainable cost model capable of achieving state-of-the-art results. The experimental findings highlight that the proposed framework delivers outstanding performance across various workloads, demonstrating advancements in cost estimation precision, plan selection robustness, and explainability, contributing to both theoretical advancements and practical improvements in database management systems.

Chapter 7

Conclusion and Future Work

In this chapter, we summarize the research work and contributions of the thesis in Section 7.1 and discuss potential avenues for future work in Section 7.2.

7.1 Conclusion

This research focuses on advancing learning-based cost models for query optimizers by addressing key challenges in cost estimation, robustness, and explainability. We conduct a comprehensive comparative study of existing state-of-the-art tree models used in cost models, evaluating their performance across various tasks and complex workloads. In parallel, we introduce Graph Neural Network (GNN) technology to the domain of tree-based query models for the first time. Specifically, we develop an innovative model, BiGG, which integrates bidirectional GNNs with Gated Recurrent Units (GRUs). Our experimental results demonstrate that BiGG significantly outperforms existing tree models in its ability to effectively represent and encode complex query plans, marking a notable advancement in the field.

Building on the foundation of the BiGG model, we propose a novel ranking-to-learn, uncertainty-aware cost model. This model introduces a two-branch estimator that simultaneously predicts query plan costs and quantifies the uncertainty associated with these predictions. To enhance decision-making, the model leverages a ranking loss function, enabling it to learn how to optimally integrate cost estimates with their uncertainties for more effective plan comparison and selection. Experimental evaluations show that this uncertainty-aware approach not only maintains superior cost estimation accuracy but also significantly improves the robustness of the model in selecting plans and managing total runtime under various workload conditions.

For the first time, we introduce explainability techniques to further enhance the trustworthiness and transparency of learning-based cost models. Our proposed explainability technique is compatible with any learning-based cost model and offers insights into the decision-making process by identifying influential subgraphs within the query plan that

impact cost predictions. This method not only enhances the explainability and credibility of the model but also leads to further improvements in cost estimation performance, creating a virtuous cycle between accuracy and transparency.

By integrating these three key components: a novel tree model BiGG leveraging bidirectional GNNs, an uncertainty-aware cost model with ranking-to-learn mechanisms and a learning-based explainability technique, we present a novel architecture Reqo that delivers comprehensive improvements across multiple dimensions. Reqo achieves superior performance in cost estimation accuracy, demonstrates enhanced robustness in plan selection, and provides clear and accurate explanations for predictions. These contributions collectively advance the state of the art, setting a new benchmark for developing learning-based query optimizers in modern database systems.

7.2 Future Work

While the cost model framework we proposed has demonstrated substantial performance improvements in various aspects of query optimization, there remain several areas that can be further improved in future research.

Advancing the tree model with specialized GNN layers. Although our BiGG model demonstrates superior capabilities in representing query plans and successfully introduces GNN technology into query plan cost estimation, it leverages existing advanced GNN methods. BiGG is an innovative application of these methods rather than a fundamental breakthrough in GNN technology itself. Future research could focus on developing customized message-passing and aggregation layers specifically tailored for query plans. Building on the strengths of existing GNN frameworks, such a specialized approach could unlock the next level of performance in tree models by better capturing the unique structural patterns of query plans.

Extending uncertainty quantification and utilization. Currently, our framework primarily focuses on quantifying uncertainty related to the input data, such as uncertainty within query plans. However, it does not account for uncertainties within the parameters of the two-branch cost model, which are used to estimate both costs and uncertainties. A valuable future direction would involve developing methods for the model to autonomously quantify a broader range of uncertainties during training, such as parameter uncertainties. Achieving more comprehensive uncertainty estimation would enable the model to provide more precise and reliable cost assessments, improving plan selection decisions in uncertain environments.

Exploring advanced integration of cost and uncertainty. Our current approach uses a learning-based adaptive MLP-based integrator to integrate quantified uncertainty and estimated cost through direct numerical summation. While effective, this method risks information loss during the integration process. Future research could explore more sophisticated integration techniques—such as multifactorial models or methods that retain the structural properties of uncertainty and cost without reducing them to numerical form.

These approaches could maximize the utilization of uncertainty and cost assessments, further enhancing the model's robustness in plan selection.

Enhancing explainability techniques. As the first study introducing explainability into learning-based cost models, our proposed technique demonstrates significant potential, but there are areas for refinement. The current approach estimates the contribution of subgraphs within query plans to the final cost prediction. However, the accuracy of these explanations is dependent on the performance of the underlying cost model and requires intermediate data—such as execution costs for various subtrees—during query execution. This increases training time and computational overhead. Additionally, the need to generate embeddings for subtrees during inference leads to a notable increase in inference time when explainability is combined with the cost model. Developing more efficient explanation methods that minimize this overhead while maintaining accuracy will be an important future challenge.

Refining structural explanations at a finer granularity. Currently, our framework does not fully elucidate the influence of specific query plan structures on cost estimation, such as how particular subgraph structures, join orders, or predicates impact the final cost. Future work could focus on refining the architecture to offer more granular explanations by detailing the influence of these components. This would enable the model to provide more actionable insights into query plans, enhancing both its explainability and utility.

Leveraging explanations for query optimization. While our explainability framework helps users and researchers understand how subgraphs contribute to costs, the next step is to translate these insights into practical optimization strategies. Future research could explore how to apply these explanations to optimize join orders or rewrite queries. Achieving this would bring explainability technology closer to real-world applications, improving query optimization outcomes through a feedback loop between cost estimation and query rewriting.

Integrating explainability with uncertainty quantification. A promising research direction involves the integration of explainability with uncertainty quantification. For example, during uncertainty estimation, the model could identify which subgraph structures or query components contribute most to uncertainty. This integration would allow the system to refine uncertainty estimates through explainable subgraphs, providing deeper insights into both prediction reliability and plan robustness. Merging these two areas could enhance both explainability and robustness, resulting in more transparent and defensible decision-making in query optimization. This unified framework would not only improve the uncertainty quantification mechanism but also lead to more efficient, reliable, and interpretable database management systems.

References

[1] Mehmet Aytimur, Silvan Reiner, Leonard Wörteler, Theodoros Chondrogiannis, and Michael Grossniklaus. Lplm: A neural language model for cardinality estimation of like-queries. Proceedings of the ACM on Management of Data, 2(1):1–25, 2024.

[2] 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.

[3] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. arXiv preprint arXiv:2102.11533, 2021.

[4] Mohit Bajaj, Lingyang Chu, Zi Yu Xue, Jian Pei, Lanjun Wang, Peter Cho-Ho Lam, and Yong Zhang. Robust counterfactual explanations on graph neural networks. Advances in Neural Information Processing Systems, 34:5644–5655, 2021.

[5] Federico Baldassarre and Hossein Azizpour. Explainability techniques for graph convolutional networks. arXiv preprint arXiv:1905.13686, 2019.

[6] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? arXiv preprint arXiv:2105.14491, 2021.

[7] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, 4(1):1–43, 2012.

[8] 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, pages 211–222, 2001.

[9] David Buterez, Jon Paul Janet, Steven J Kiddle, Dino Oglic, and Pietro Liò. Graph neural networks with adaptive readouts. Advances in Neural Information Processing Systems, 35:19746–19758, 2022.

[10] Walter Cai, Magdalena Balazinska, and Dan Suciu. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In Proceedings of the 2019 International Conference on Management of Data, pages 18–35, 2019.

[11] Jin Chen, Guanyu Ye, Yan Zhao, Shuncheng Liu, Liwei Deng, Xu Chen, Rui Zhou, and Kai Zheng. Efficient join order selection learning with graph-based representation. In Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining, pages 97–107, 2022.

[12] 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.

[13] 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.

[14] Tingyang Chen, Dazhuo Qiu, Yinghui Wu, Arijit Khan, Xiangyu Ke, and Yunjun Gao. View-based explanations for graph neural networks. arXiv preprint arXiv:2401.02086, 2024.

[15] 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. Proceedings of the VLDB Endowment, 16(9):2261–2273, 2023.

[16] Xu Chen, Zhen Wang, Shuncheng Liu, Yaliang Li, Kai Zeng, Bolin Ding, Jingren Zhou, Han Su, and Kai Zheng. Base: Bridging the gap between cost and latency for query optimization. Proceedings of the VLDB Endowment, 16(8):1958–1966, 2023.

[17] Ziheng Chen, Fabrizio Silvestri, Jia Wang, Yongfeng Zhang, Zhenhua Huang, Hongshik Ahn, and Gabriele Tolomei. Grease: Generate factual and counterfactual explanations for gnn-based recommendations. arXiv preprint arXiv:2208.04222, 2022.

[18] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[19] Stavros Christodoulakis. Implications of certain assumptions in database performance evauation. ACM Transactions on Database Systems (TODS), 9(2):163–186, 1984.

[20] 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, pages 293–302, 2002.

[21] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. Advances in Neural Information Processing Systems, 33:13260–13271, 2020.

[22] Enyan Dai and Suhang Wang. Towards self-explainable graph neural network. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 302–311, 2021.

[23] Amol Deshpande, Minos Garofalakis, and Rajeev Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. ACM SIGMOD Record, 30(2):199–210, 2001.

[24] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R Narasayya. Ai meets ai: Leveraging query executions to improve index recommendations. In Proceedings of the 2019 International Conference on Management of Data, pages 1241–1258, 2019.

[25] Monroe D Donsker and SR Srinivasa Varadhan. Asymptotic evaluation of certain markov process expectations for large time, i. Communications on pure and applied mathematics, 28(1):1–47, 1975.

[26] 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):1–25, 2023.

[27] 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):1–25, 2023.

[28] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In Algorithms-ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings 11, pages 605–617. Springer, 2003.

[29] Anshuman Dutt and Jayant R Haritsa. Plan bouquets: A fragrant approach to robust query processing. ACM Transactions on Database Systems (TODS), 41(2):1–37, 2016.

[30] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. Selectivity estimation for range predicates using lightweight models. Proceedings of the VLDB Endowment, 12(9):1044–1057, 2019.

[31] Aosong Feng, Chenyu You, Shiqiang Wang, and Leandros Tassiulas. Kergnns: Interpretable graph neural networks with graph kernels. In Proceedings of the AAAI conference on artificial intelligence, volume 36, pages 6614–6622, 2022.

[32] Qizhang Feng, Ninghao Liu, Fan Yang, Ruixiang Tang, Mengnan Du, and Xia Hu. Degree: Decomposition based explanation for graph neural networks. arXiv preprint arXiv:2305.12895, 2023.

[33] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. Discrete mathematics & theoretical computer science, (Proceedings), 2007.

[34] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. Journal of computer and system sciences, 31(2):182–209, 1985.

[35] Dennis Fuchs, Zhen He, and Byung Suk Lee. Compressed histograms with arbitrary bucket layouts for selectivity estimation. Information Sciences, 177(3):680–702, 2007.

[36] Thorben Funke, Megha Khosla, Mandeep Rathee, and Avishek Anand. Z orro: Valid, sparse, and stable explanations in graph neural networks. IEEE Transactions on Knowledge and Data Engineering, 2022.

[37] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning, pages 1050–1059. PMLR, 2016.

[38] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In Proceedings of the 2001 ACM SIGMOD international conference on Management of data, pages 461–472, 2001.

[39] Frédéric Giroire. Order statistics and estimating cardinalities of massive data sets. Discrete Applied Mathematics, 157(2):406–427, 2009.

[40] Dimitrios Gunopulos, George Kollios, Vassilis J Tsotras, and Carlotta Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. Acm Sigmod Record, 29(2):463–474, 2000.

[41] 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.

[42] Gérard Hamiache and Florian Navarro. Associated consistency, value and graphs. International Journal of Game Theory, 49:227–249, 2020.

[43] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. Advances in neural information processing systems, 30, 2017.

[44] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. Advances in neural information processing systems, 30, 2017.

[45] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, et al. Cardinality estimation in dbms: A comprehensive benchmark evaluation. arXiv preprint arXiv:2109.05877, 2021.

[46] Jayant R Haritsa. Robust query processing: Mission possible. In 2019 IEEE 35th International Conference on Data Engineering (ICDE), pages 2072–2075. IEEE, 2019.

[47] 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, pages 1477–1492, 2015.

[48] 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, pages 697–709, 2021.

[49] Benjamin Hilprecht and Carsten Binnig. Zero-shot cost models for out-of-the-box learned cost prediction. arXiv preprint arXiv:2201.00561, 2022.

[50] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. Deepdb: Learn from data, not from queries! arXiv preprint arXiv:1909.00607, 2019.

[51] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Comput., 9(8):1735–1780, nov 1997.

[52] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. arXiv preprint arXiv:1905.12265, 2019.

[53] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. Graphlime: Local interpretable model explanations for graph neural networks. IEEE Transactions on Knowledge and Data Engineering, 2022.

[54] Zexi Huang, Mert Kosan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Global counterfactual explainer for graph neural networks. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, pages 141–149, 2023.

[55] Zexi Huang, Mert Kosan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Global counterfactual explainer for graph neural networks. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, pages 141–149, 2023.

[56] Yannis Ioannidis. The history of histograms (abridged). In Proceedings 2003 VLDB Conference, pages 19–30. Elsevier, 2003.

[57] Jaykumar Kakkad, Jaspal Jannu, Kartik Sharma, Charu Aggarwal, and Sourav Medya. A survey on explainability of graph neural networks. arXiv preprint arXiv:2306.01958, 2023.

[58] Amin Kamali, Verena Kantere, Calisto Zuzarte, and Vincent Corvinelli. Roq: Robust query optimization based on a risk-aware learned cost model. arXiv preprint arXiv:2401.15210, 2024.

[59] 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.

[60] Srinivas Karthik, Jayant R Haritsa, Sreyash Kenkre, Vinayaka Pandit, and Lohit Krishnan. Platform-independent robust query processing. IEEE Transactions on Knowledge and Data Engineering, 31(1):17–31, 2017.

[61] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. Estimating join selectivities using bandwidth-optimized kernel density models. Proceedings of the VLDB Endowment, 10(13):2085–2096, 2017.

[62] Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. arXiv preprint arXiv:2204.04879, 2022.

[63] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[64] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[65] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. Physical review E, 69(6):066138, 2004.

[66] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. Learning to optimize join queries with deep reinforcement learning. arXiv preprint arXiv:1808.03196, 2018.

[67] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. Advances in neural information processing systems, 30, 2017.

[68] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. arXiv preprint arXiv:1711.00165, 2017.

[69] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In International conference on machine learning, pages 3734–3743. PMLR, 2019.

[70] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? Proceedings of the VLDB Endowment, 9(3):204–215, 2015.

[71] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. Cardinality estimation done right: Index-based join sampling. In Cidr, 2017.

[72] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In Proceedings of the 2016 International Conference on Management of Data, pages 615–629, 2016.

[73] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. arXiv preprint arXiv:2006.07739, 2020.

[74] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. Proceedings of Machine Learning and Systems, 2:230–246, 2020.

[75] Wenqian Li, Yinchuan Li, Zhigang Li, Jianye Hao, and Yan Pang. Dag matters! gflownets enhanced explainer for graph neural networks. arXiv preprint arXiv:2303.02448, 2023.

[76] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In International conference on machine learning, pages 3835–3845. PMLR, 2019.

[77] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. arXiv preprint arXiv:1807.05118, 2018.

[78] Wanyu Lin, Hao Lan, and Baochun Li. Generative causal explanations for graph neural networks. In International Conference on Machine Learning, pages 6666–6679. PMLR, 2021.

[79] Richard J Lipton, Jeffrey F Naughton, and Donovan A Schneider. Practical selectivity estimation through adaptive sampling. In Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pages 1–11, 1990.

[80] Feilong Liu and Spyros Blanas. Forecasting the cost of processing multi-join queries via hashing for main-memory databases. In Proceedings of the Sixth ACM Symposium on Cloud Computing, pages 153–166, 2015.

[81] Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. Advances in neural information processing systems, 33:7498–7512, 2020.

[82] Jie Liu, Wenqian Dong, Qingqing Zhou, and Dong Li. Fauce: Fast and accurate deep ensembles with uncertainty for cardinality estimation. Proceedings of the VLDB Endowment, 14(11):1950–1963, 2021.

[83] Shuncheng Liu, Xu Chen, Yan Zhao, Jin Chen, Rui Zhou, and Kai Zheng. Efficient learning with pseudo labels for query cost estimation. In Proceedings of the 31st ACM International Conference on Information & Knowledge Management, pages 1309–1318, 2022.

[84] Ana Lucic, Maartje A Ter Hoeve, Gabriele Tolomei, Maarten De Rijke, and Fabrizio Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks.

In International Conference on Artificial Intelligence and Statistics, pages 4499–4511. PMLR, 2022.

[85] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. Advances in neural information processing systems, 33:19620–19631, 2020.

[86] Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. A flexible generative framework for graph-based semi-supervised learning. Advances in Neural Information Processing Systems, 32, 2019.

[87] Jing Ma, Ruocheng Guo, Saumitra Mishra, Aidong Zhang, and Jundong Li. Clear: Generative counterfactual explanations on graphs. Advances in neural information processing systems, 35:25895–25907, 2022.

[88] 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, pages 1275–1288, 2021.

[89] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. arXiv preprint arXiv:1904.03711, 2019.

[90] 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, pages 1–4, 2018.

[91] Ryan Marcus and Olga Papaemmanouil. Plan-structured deep neural network models for query performance prediction. arXiv preprint arXiv:1902.00132, 2019.

[92] Volker Markl, Vijayshankar Raman, David Simmen, Guy Lohman, Hamid Pirahesh, and Miso Cilimdzic. Robust query processing through progressive optimization. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pages 659–670, 2004.

[93] Siqi Miao, Mia Liu, and Pan Li. Interpretable and generalizable graph learning via stochastic attention mechanism. In International Conference on Machine Learning, pages 15524–15543. PMLR, 2022.

[94] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[95] 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 AAAI conference on artificial intelligence, volume 30, 2016.

[96] Chiraz Moumen, Franck Morvan, and Abdelkader Hameurlain. Handling estimation inaccuracy in query optimization. In Web Technologies and Applications: 18th Asia-Pacific Web Conference, APWeb 2016, Suzhou, China, September 23-25, 2016. Proceedings, Part II, pages 355–367. Springer, 2016.

[97] M Muralikrishna and David J DeWitt. Equi-depth multidimensional histograms. In Proceedings of the 1988 ACM SIGMOD international conference on Management of data, pages 28–36, 1988.

[98] 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. Proceedings of the VLDB Endowment, 16(6):1520–1533, 2023.

[99] AkshatKumar Nigam, Robert Pollice, Mario Krenn, Gabriel dos Passos Gomes, and Alan Aspuru-Guzik. Beyond generative models: superfast traversal, optimization, novelty, exploration and discovery (stoned) algorithm for molecules using selfies. Chemical science, 12(20):7079–7090, 2021.

[100] David A Nix and Andreas 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. IEEE, 1994.

[101] Danilo Numeroso and Davide Bacciu. Meg: Generating molecular counterfactual explanations for deep graph networks. In 2021 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2021.

[102] Frank Olken and Doron Rotem. Random sampling from database files: A survey. In International Conference on Scientific and Statistical Database Management, pages 92–111. Springer, 1990.

[103] Yeonsu Park, Seongyun Ko, Sourav S Bhowmick, Kyoungmin Kim, Kijae Hong, and Wook-Shin Han. G-care: A framework for performance benchmarking of cardinality estimation techniques for subgraph matching. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pages 1099–1114, 2020.

[104] Emanuel Parzen. On estimation of a probability density function and mode. The annals of mathematical statistics, 33(3):1065–1076, 1962.

[105] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative styl, high-performance deep learning library. Advances in neural information processing systems, 32, 2019.

[106] Robin Pemantle. Vertex-reinforced random walk. Probability Theory and Related Fields, 92(1):117–136, 1992.

[107] Tamara Pereira, Erik Nascimento, Lucas E Resck, Diego Mesquita, and Amauri Souza. Distill n'explain: explaining graph neural networks using simple surrogates. In International Conference on Artificial Intelligence and Statistics, pages 6199–6214. PMLR, 2023.

[108] Meikel Poess and Chris Floyd. New tpc benchmarks for decision support and web commerce. ACM Sigmod Record, 29(4):64–71, 2000.

[109] Meikel Poess, Raghunath Othayoth Nambiar, and David Walrath. Why you should run tpc-ds: A workload analysis. In VLDB, volume 7, pages 1138–1149, 2007.

[110] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pages 689–690. IEEE, 2011.

[111] Viswanath Poosala and Yannis E Ioannidis. Selectivity estimation without the attribute value independence assumption. In VLDB, volume 97, pages 486–495, 1997.

[112] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10772–10781, 2019.

[113] PostgreSQL Documentation 15. Chapter 75.1 row estimation examples, 2023.

[114] Emanuele Rossi, Bertrand Charpentier, Francesco Di Giovanni, Fabrizio Frasca, Stephan Günnemann, and Michael Bronstein. Edge directionality improves learning on heterophilic graphs. arXiv preprint arXiv:2305.10498, 2023.

[115] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B Wiltschko. A gentle introduction to graph neural networks. Distill, 6(9):e33, 2021.

[116] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting graph neural networks for nlp with differentiable edge masking. arXiv preprint arXiv:2010.00577, 2020.

[117] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof T Schütt, Klaus-Robert Müller, and Grégoire Montavon. Higher-order explanations of graph neural networks via relevant walks. IEEE transactions on pattern analysis and machine intelligence, 44(11):7581–7596, 2021.

[118] Caihua Shan, Yifei Shen, Yao Zhang, Xiang Li, and Dongsheng Li. Reinforcement learning enhanced explainer for graph neural networks. Advances in Neural Information Processing Systems, 34:22523–22533, 2021.

[119] Simon J Sheather and Michael C Jones. A reliable data-based bandwidth selection method for kernel density estimation. Journal of the Royal Statistical Society: Series B (Methodological), 53(3):683–690, 1991.

[120] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. Journal of Machine Learning Research, 12(9), 2011.

[121] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. arXiv preprint arXiv:2009.03509, 2020.

[122] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. Advances in neural information processing systems, 29, 2016.

[123] Utkarsh Srivastava, Peter J Haas, Volker Markl, Marcel Kutsch, and Tam Minh Tran. Isomer: Consistent histogram construction using query feedback. In 22nd International Conference on Data Engineering (ICDE'06), pages 39–39. IEEE, 2006.

[124] Michael Stillger, Guy M Lohman, Volker Markl, and Mokhtar Kandil. Leo-db2's learning optimizer. In VLDB, volume 1, pages 19–28, 2001.

[125] Ji Sun and Guoliang Li. An end-to-end learning-based cost estimator. arXiv preprint arXiv:1906.02560, 2019.

[126] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv:1503.00075, 2015.

[127] Shyam A Tailor, Felix L Opolka, Pietro Lio, and Nicholas D Lane. Do we need anisotropic graph neural networks? arXiv preprint arXiv:2104.01481, 2021.

[128] Juntao Tan, Shijie Geng, Zuohui Fu, Yingqiang Ge, Shuyuan Xu, Yunqi Li, and Yongfeng Zhang. Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. In Proceedings of the ACM Web Conference 2022, pages 1018–1027, 2022.

[129] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika, 25(3-4):285–294, 1933.

[130] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In 2015 ieee information theory workshop (itw), pages 1–5. IEEE, 2015.

[131] Kostas Tzoumas, Amol Deshpande, and Christian S Jensen. Lightweight graphical models for selectivity estimation without independence assumptions. Proceedings of the VLDB Endowment, 4(11):852–863, 2011.

[132] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[133] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. arXiv preprint arXiv:1710.10903, 2017.

[134] Minh Vu and My T Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. Advances in neural information processing systems, 33:12225–12235, 2020.

[135] Hai Wang and Kenneth C Sevcik. A multi-dimensional histogram for selectivity estimation and fast approximate query answering. In Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, pages 328–342, 2003.

[136] 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.

[137] Jiayi Wang, Chengliang Chai, Jiabin Liu, and Guoliang Li. Cardinality estimation using normalizing flow. The VLDB Journal, pages 1–26, 2023.

[138] Peihao Wang, Yuehao Wang, Hua Lin, and Jianbo Shi. Sogcn: Second-order graph convolutional networks. arXiv preprint arXiv:2110.07141, 2021.

[139] Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. Am-gcn: Adaptive multi-channel graph convolutional networks. In Proceedings of the 26th ACM SIGKDD International conference on knowledge discovery & data mining, pages 1243–1253, 2020.

[140] Xiaoqi Wang and Han-Wei Shen. Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks. arXiv preprint arXiv:2209.07924, 2022.

[141] Zilong Wang, Qixiong Zeng, Ning Wang, Haowen Lu, and Yue Zhang. Ceda: Learned cardinality estimation with domain adaptation. Proceedings of the VLDB Endowment, 16(12):3934–3937, 2023.

[142] Geemi P Wellawatte, Aditi Seshadri, and Andrew D White. Model agnostic generation of counterfactual explanations for molecules. Chemical science, 13(13):3697–3705, 2022.

[143] Kyu-Young Whang, Brad T Vander-Zanden, and Howard M Taylor. A linear-time probabilistic counting algorithm for database applications. ACM Transactions on Database Systems (TODS), 15(2):208–229, 1990.

[144] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. Adaptation, learning, and optimization, 12(3):729, 2012.

[145] Florian Wolf, Michael Brendle, Norman May, Paul R Willems, Kai-Uwe Sattler, and Michael Grossniklaus. Robustness metrics for relational query execution plans. 2018.

[146] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In International conference on machine learning, pages 6861–6871. PMLR, 2019.

[147] Wentao Wu, Yun Chi, Shenghuo Zhu, Junichi Tatemura, Hakan Hacigümüs, and Jeffrey F Naughton. Predicting query execution time: Are optimizer cost models really unusable? In 2013 IEEE 29th International Conference on Data Engineering (ICDE), pages 1081–1092. IEEE, 2013.

[148] Wentao Wu, Jeffrey F Naughton, and Harneet Singh. Sampling-based query re-optimization. In Proceedings of the 2016 International Conference on Management of Data, pages 1721–1736, 2016.

[149] Ying-Xin Wu, Xiang Wang, An Zhang, Xiangnan He, and Tat-Seng Chua. Discovering invariant rationales for graph neural networks. arXiv preprint arXiv:2201.12872, 2022.

[150] Ziniu Wu, Amir Shaikhha, Rong Zhu, Kai Zeng, Yuxing Han, and Jingren Zhou. Bayescard: Revitilizing bayesian frameworks for cardinality estimation, 2021.

[151] 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.

[152] Ziniu Wu, Rong Zhu, Andreas Pfadler, Yuxing Han, Jiangneng Li, Zhengping Qian, Kai Zeng, and Jingren Zhou. Fspn: A new class of probabilistic graphical model. arXiv preprint arXiv:2011.09020, 2020.

[153] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. IEEE transactions on pattern analysis and machine intelligence, 41(9):2251–2265, 2018.

[154] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826, 2018.

[155] 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, pages 931–944, 2022.

[156] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. Neurocard: one cardinality estimator for all tables. arXiv preprint arXiv:2006.08109, 2020.

[157] 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. arXiv preprint arXiv:1905.04278, 2019.

[158] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. Advances in neural information processing systems, 32, 2019.

[159] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. Advances in neural information processing systems, 31, 2018.

[160] Junchi Yu, Jie Cao, and Ran He. Improving subgraph recognition with variational graph information bottleneck. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 19396–19405, 2022.

[161] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. Graph information bottleneck for subgraph recognition. arXiv preprint arXiv:2010.05563, 2020.

[162] 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.

[163] 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. IEEE, 2020.

[164] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. Automatic view generation with deep learning and reinforcement learning. In 2020 IEEE 36th International Conference on Data Engineering (ICDE), pages 1501–1512. IEEE, 2020.

[165] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pages 430–438, 2020.

[166] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In International conference on machine learning, pages 12241–12252. PMLR, 2021.

[167] Ji Zhang, K Zhou, and S Schelter. Alphajoin: Join order selection à la alphago. PhD@ VLDB, 2652, 2020.

[168] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.

[169] Shichang Zhang, Yozen Liu, Neil Shah, and Yizhou Sun. Gstarx: Explaining graph neural networks with structure-aware cooperative games. Advances in Neural Information Processing Systems, 35:19810–19823, 2022.

[170] Yue Zhang, David Defazio, and Arti Ramesh. Relex: A model-agnostic relational model explainer. In Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society, pages 1042–1049, 2021.

[171] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. Protgnn: Towards self-explaining graph neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 9127–9135, 2022.

[172] Kangfei Zhao, Jeffrey Xu Yu, Zongyan He, Rui Li, and Hao Zhang. Lightweight and accurate cardinality estimation by neural network gaussian process. In Proceedings of the 2022 International Conference on Management of Data, pages 973–987, 2022.

[173] 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.

[174] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. Queryformer: A tree transformer model for query plan representation. Proceedings of the VLDB Endowment, 15(8):1658–1670, 2022.

[175] Yue Zhao, Zhaodonghui Li, and Gao Cong. A comparative study and component analysis of query plan representation techniques in ml4db studies. Proceedings of the VLDB Endowment, 17(4):823–835, 2023.

[176] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In Proceedings of the 2018 International Conference on Management of Data, pages 1525–1539, 2018.

[177] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. AI open, 1:57–81, 2020.

[178] Qi Zhou, Joy Arulraj, Shamkant Navathe, William Harris, and Jinpeng Wu. Sia: Optimizing queries using learned predicates. In Proceedings of the 2021 International Conference on Management of Data, pages 2169–2181, 2021.

[179] Weiqing Zhou, Siyu Zhan, Bo Dai, and Lei Guo. Soar: A learned join order selector with graph attention mechanism. In 2022 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2022.

[180] Xuanhe Zhou, Guoliang Li, Chengliang Chai, and Jianhua Feng. A learned query rewrite system using monte carlo tree search. Proceedings of the VLDB Endowment, 15(1):46–58, 2021.

[181] Xuanhe Zhou, Guoliang Li, Jianming Wu, Jiesi Liu, Zhaoyan Sun, and Xinning Zhang. A learned query rewrite system. Proceedings of the VLDB Endowment, 16(12):4110–4113, 2023.

[182] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. Lero: A learning-to-rank query optimizer. Proceedings of the VLDB Endowment, 16(6):1466–1479, 2023.

[183] 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. arXiv preprint arXiv:2011.09022, 2020.

[184] Xiaofeng Zhu, Shichao Zhang, Yonggang Li, Jilian Zhang, Lifeng Yang, and Yue Fang. Low-rank sparse subspace for spectral clustering. IEEE Transactions on knowledge and data engineering, 31(8):1532–1543, 2018.