# Cross-Database Query Cost Estimation:

# A Comparative Study of Classic ML, Transformers, and LLMs

**Junran Jia (Jasmine)**

April 2025

Supervisor: Prof. Lin Ma

In partial fulfillment of the requirements for the degree of
*Bachelor of Science in Data Science (Honors)*
Department of Statistics

# ABSTRACT

Optimizing query performance is a foundational challenge in modern database systems. While machine learning for databases (ML4DB) has demonstrated strong in-system prediction capabilities, most models remain tightly coupled to specific database engines, limiting their cross-system generalizability. This thesis explores whether query cost estimation models can be designed to generalize across heterogeneous DBMSs, such as PostgreSQL and MySQL, without requiring extensive retraining. We evaluate three modeling approaches: (1) a classic operator-level machine learning (ML) method, (2) QueryFormer, a tree-structured Transformer model, and (3) large language models (LLMs) fine-tuned for structured query tasks. To bridge domain gaps in query cost estimation across heterogeneous database systems, we apply a stacked model approach for Classic ML and QueryFormer. In contrast, LLMs are evaluated using a direct transfer strategy, as their semantic embeddings enable cross-system generalization without the need for retraining. Using the TPC-H benchmark, we compare these models on regression and classification metrics, including $q$-error, Mean Relative Error (MRE), accuracy, and Mean Bucket Distance (MBD). Our results show that LLMs achieve the highest accuracy and generalization with minimal target data, but offer limited gains from cross-database transfer and incur high computational costs. QueryFormer strikes an effective balance, benefiting significantly from stacking and maintaining consistent performance across metrics and database systems. Classic ML, while efficient, struggles with domain shifts and sometimes degrades under stacking. These findings highlight the importance of model architecture and transfer strategy in ML4DB tasks and suggest future directions toward hybrid and adaptive models for robust cross-database learning.

# TABLE OF CONTENTS

CHAPTER

# CHAPTER 1

# Introduction

Optimizing database performance has long been a critical task in the realm of data management, with machine learning for databases (ML4DB) emerging as a transformative approach. Tasks such as cost estimation, cardinality prediction, and query optimization are essential to enhance database efficiency, affecting execution times and resource utilization.

Existing ML4DB models are typically trained on a single database system, limiting their ability to generalize between different systems. This dependency not only incurs high computational costs but also creates inefficiencies when transitioning models between systems. Lack of cross-system adaptability often results in redundant retraining, increasing both the cost and the complexity of maintaining optimal performance [16]. This challenge is further highlighted by recent work showing that these instance-specific models require retraining whenever the DBMS environment changes, with over 93% of time spent on running queries for training rather than tuning [7].

Although large language models (LLMs) have demonstrated remarkable generalization capabilities in diverse fields, their performance in database tasks remains suboptimal. LLMs are not inherently specialized for database operations and often struggle with tasks that require domain-specific optimizations, such as cost or cardinality estimation. This limitation is consistent with broader findings on the weaknesses of LLMs in engineering-related tasks, where they are unable to provide high-quality AI-based tools due to gaps in specialized knowledge [11]. This gap underscores the need for specialized models tailored to database tasks.

This study investigates whether a specialized database model can achieve strong cross-system performance without requiring extensive retraining on different database environ-

ments. Addressing this challenge could transform ML4DB by reducing system dependency and enhancing scalability.

To this end, we explore three modeling approaches—classic machine learning (ML) methods, QueryFormer (QF), and LLM-based approaches—focusing on how different techniques can facilitate built-in database statistics and optimizer estimations. Our primary goal is to develop and compare methods for training models that can perform cross-database query runtime or cost estimation, a crucial step toward building a more generalized foundation model for database optimization.

Using the TPC-H benchmark, we evaluated these approaches on the basis of their generalizability, predictive accuracy, and overall effectiveness in database performance modeling. Additionally, we investigate the trade-offs between regression- and classification-based training for query runtime prediction, assessing classification models using accuracy and Mean Bucket Distance (MBD) and regression models using $q$-error and Mean Relative Error (MRE).

**Result Summary.** In cross-database settings, LLM-based models demonstrate strong generalization, consistently achieving the highest accuracy and lowest mean bucket distance (MBD), even with minimal target data. However, their performance gains from stacking are limited, and they incur substantial computational costs. QueryFormer proves to be the most effective for cross-database knowledge transfer, showing stable improvements from stacking and balanced performance across metrics. In contrast, Classic ML models struggle to adapt across database boundaries due to their reliance on static features and operator-level predictions, and often see degraded performance when stacking is applied. These results emphasize the critical role of structural representation and adaptability in cross-DB transfer and motivate future exploration of hybrid models tailored for heterogeneous database environments.

# CHAPTER 2

# Background

## 2.1 ML4DB Methods

Machine Learning for Databases (ML4DB) aims to improve core DBMS tasks using machine learning techniques. As summarized by Cong et al.[1], the field can be organized into three major themes: foundations, paradigms, and open problems. Foundations refer to common techniques like query plan representation and transfer learning models. Paradigms distinguish between ML as a replacement versus an enhancement to existing database components [8, 9, 10]. Open problems include model efficiency, handling data shifts, and building general-purpose models that transfer across tasks [1].

This thesis focuses on the *cost estimation* task, which predicts the latency or resource consumption of a query plan—a critical step for query optimizers to choose efficient execution strategies. Traditional cost models are built on heuristics and statistical assumptions, often breaking down on complex or correlated data.

Over time, ML-based cost estimation models have evolved from early feature-based regressors(e.g., MSCN [4]) to tree-structured neural networks like Plan-Cost [10] and E2E-Cost [13], and more recently to models that deeply encode query semantics. Among these, BAO [8], which enhances existing optimizers via reinforcement learning, and QueryFormer [15], a transformer-based model for general-purpose query representation, have emerged as state-of-the-art. A comparative study by Zhao et al.[16] confirmed that both BAO and QueryFormer consistently achieve top-tier performance across in-distribution and out-of-distribution workloads.

Despite these advances, most ML4DB models are trained on a single database system, making them tightly coupled to system-specific query plans and statistics. The predominant

paradigm, instance-specific learning, involves executing large sets of SQL queries on each target database to gather training data. While this can exploit system-specific patterns, it incurs high computational costs and requires retraining for each new system. Moreover, many models rely on manually engineered features tied to a specific optimizer, limiting portability and scalability [16].

In response, recent work on foundation database models proposes to learn transferable representations across tasks and datasets using pre-trained components (e.g., logical plan or data experts) [16]. However, these models still require retraining when applied across different database systems, limiting their practical generalization.

In contrast, our goal is to develop a cross-system foundation model that generalizes across tasks, datasets, and DBMS architectures (e.g., PostgreSQL, MySQL, DuckDB) without system-specific retraining. This would significantly improve scalability and reduce the engineering effort required to deploy ML models in diverse real-world database environments.

## 2.2 Using LLMs in Databases

While Large Language Models (LLMs) have shown impressive generalization in NLP and vision-language domains, their application to structured database tasks like cost estimation remains limited. General-purpose LLMs (E.g., GPT, LLaMA [12, 2]) lack built-in mechanisms to interpret critical database statistics—such as histograms, indexes, and execution plans—that are essential for accurate query performance prediction. Their outputs are often brittle—small changes in prompts can yield large variances in predictions—and they may suffer from hallucinations, generating plausible but incorrect answers [3].

Moreover, general-purpose LLMs tend to struggle with mathematical reasoning, a key requirement in database optimization. Unlike natural language tasks, database reasoning involves precise numerical calculations, symbolic logic, and structural dependencies that are not always explicit. This makes tasks like query runtime prediction especially challenging: even small prediction errors can lead to order-of-magnitude mistakes in cost estimation [5, 6].

These limitations pose serious challenges for directly applying general-purpose LLMs to ML4DB tasks. However, their strong generalization ability and capacity to encode complex patterns make them an attractive candidate for foundational work in this space—provided

they are properly adapted. By fine-tuning LLMs with task-specific data and structured representations—such as query plans, operator trees, and execution traces—we aim to reduce their dependency on natural language patterns and ground them in semantically meaningful input. To address their weakness in mathematical reasoning, we carefully control the output format, minimize numeric variance in targets (e.g., by predicting log-scale values), and constrain prediction tasks to classification or regression with bounded outputs. Through this design, we explore whether LLMs—when structured appropriately—can learn to approximate cost models with competitive accuracy and potentially generalize across systems and workloads more flexibly than traditional ML models. This investigation serves as a step toward building interpretable, adaptable, and scalable foundation models for database systems.

# CHAPTER 3

# Method

**Chapter Overview.** This chapter outlines the methods used to evaluate query cost estimation across both same-database and cross-database settings. We consider three modeling approaches: (1) Classic Machine Learning (ML) using operator-level features and AutoML, (2) QueryFormer, a tree-structured Transformer designed for encoding execution plans, and (3) Large Language Models (LLMs) fine-tuned with structured query inputs. Section 3.1 introduces these three models and their use for same-database prediction. Section 3.2 explains how we adapt them for cross-database prediction, including a stacked model design for ML and QueryFormer, and a direct transfer approach for LLMs. Section 3.3 defines the evaluation metrics used for comparing model performance across both settings.

## 3.1 Same-Database Cost Estimation

### 3.1.1 Model 1. Classic Machine Learning

We include this Classic ML method as a simple, interpretable baseline to benchmark the performance of more advanced models. This model predicts query execution times by learning from structured features at the operator level. We train the model on raw EXPLAIN outputs (i.e., raw query plans) to predict runtimes of entire queries (obtained from EXPLAIN ANALYZE outputs), and decomposed into individual operators such as Index Scan, Hash Join, and Sort. Each operator instance is converted into a row in a flattened tabular format. Input features include:

- Plan structure: Operator type, number of children, join and scan types
- Optimizer estimates: Estimated rows and estimated costs
- Parallel features: Worker utilization, execution distribution across threads

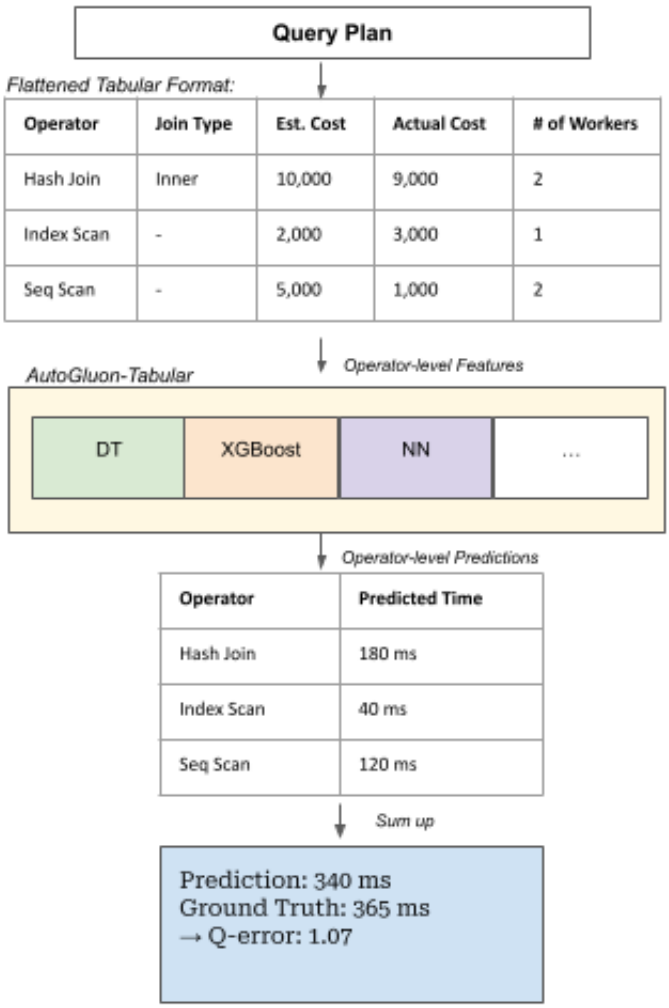


Figure 3.1: Classic ML Model Structure

The target variable is the actual operator-level execution time. No runtime labels are used as input features to prevent data leakage.

We train the model using **AutoGluon-Tabular**, an AutoML framework that ensembles diverse models using multi-layer stacking (see Figure 3.1). This setup helps improve generalization across heterogeneous query plans. The predicted execution time for a full query is computed by summing the predicted runtimes of its constituent operators. This model serves as a baseline for assessing advanced representation methods. In most cases, AutoGluon constructs the final model by combining LightGBM, CatBoost, and RandomForestMSE with varying ensemble weights.

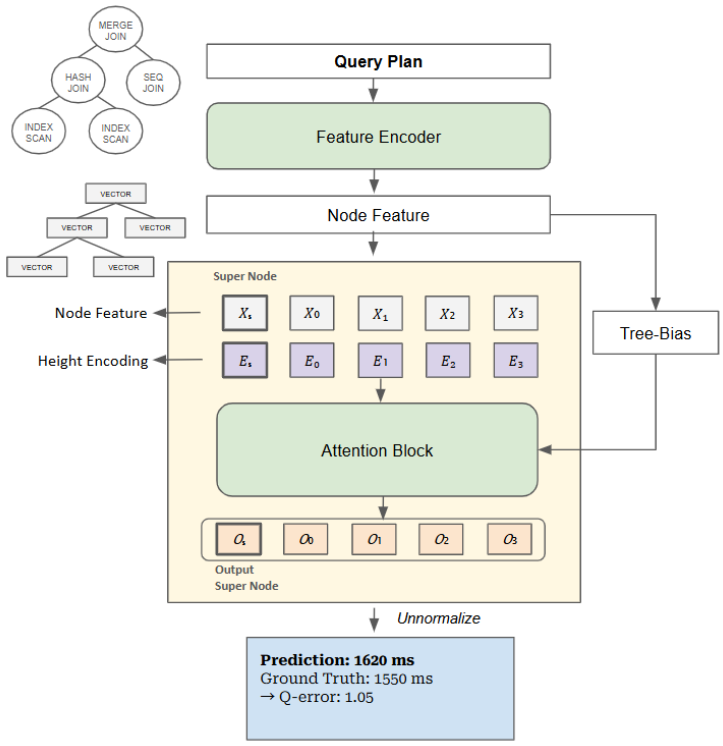


Figure 3.2: QueryFormer Model Structure

### 3.1.2 Model 2. QueryFormer

We include QueryFormer because it represents the current state-of-the-art in ML-based query cost estimation, demonstrating strong performance across both in-distribution and out-of-distribution workloads [16].

QueryFormer is a tree-structured Transformer model that learns vectorized representations of query execution plans, capturing both operator-level details and global plan semantics. Each node in a query plan—typically corresponding to a relational operator like a join or scan—is embedded using a combination of learned representations for categorical attributes (such as operator type, join conditions, and referenced tables) and numeric features that characterize data distributions (e.g., histograms and sampled tuples). This rich encoding enables the model to capture predicate selectivity and the statistical behavior of columns, which are crucial for accurate cost estimation. Unlike traditional approaches that rely on manually crafted features or optimizer-estimated cardinalities, QueryFormer directly integrates raw statistical inputs, making it more robust to variations in query structure and data skew [15, 16].

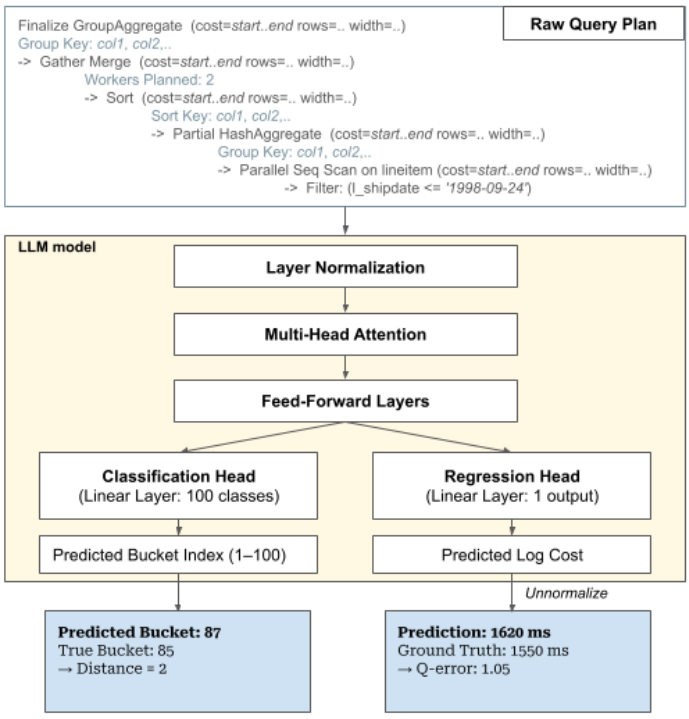


Figure 3.3: Llama Finetuning Model Structure

To effectively model the hierarchical nature of query execution plans, QueryFormer introduces several novel architectural modifications to the standard Transformer framework (see Figure 3.2. (1) The **Tree-Bias Attention** mechanism biases the self-attention computation to favor structurally valid paths, such as parent-child and ancestor-descendant relationships, which mirrors the actual execution dependencies in query trees. (2) **Height Encoding** adds positional information based on node depth in the tree, helping the model distinguish between leaf-level scans and high-level aggregation or join operations. (3) Additionally, a **Super Node** is injected into the model, acting as a global aggregator that attends to all nodes in the tree and distills overall plan semantics into a single representation for downstream prediction tasks. This design allows QueryFormer to capture both local operator interactions and global query context, enabling it to outperform flat or sequence-based models in predicting total execution time—without the need for hand-engineered features or recursive architectures that are difficult to train.

### 3.1.3 Model 3. Fine-Tuning Large Language Models (LLMs)

We choose to fine-tune LLMs for their strong generalization capabilities and ability to learn from unstructured or semi-structured data without extensive feature engineering. Specifi-

cally, we use the LLaMA 3 models because they are open-source, performant across a wide range of NLP tasks, and available in multiple sizes (e.g., 3B, 8B, 70B), making them suitable for experimentation under varying compute constraints. Their flexibility and scalability make them a promising candidate for modeling complex patterns in query execution plans.

We fine-tune the LLaMA 3.2-3B and LLaMA 3-8B models on raw EXPLAIN outputs (i.e., raw query plans) to predict runtimes of entire queries, as measured by their corresponding EXPLAIN ANALYZE outputs. As illustrated in Figure 3.3, the model architecture consists of a sequence of transformer blocks—layer normalization, multi-head self-attention, and feed-forward layers—followed by two task-specific heads. The classification head maps each query plan to one of 100 evenly distributed latency buckets, while the regression head outputs a continuous log-scale estimate of query latency, which is later unnormalized to produce runtime in milliseconds. This dual-objective setup captures both coarse-grained and fine-grained latency signals.

To balance training efficiency with representational capacity, we explore both LLaMA 3.2-3B and 8B variants. Fine-tuning on domain-specific data has been shown to enrich LLM capabilities, enhance reliability, and reduce hallucination [5, 3]. Following this approach, we investigate two predictive settings:

- **LLM Regression:** Predicts a continuous runtime directly from the query plan.
- **LLM Classification:** Discretizes runtimes into 100 latency buckets and classifies each query into the appropriate bucket.

The classification setting helps mitigate LLMs' numeric brittleness by modeling broader distributional patterns instead of relying on precise value prediction. For comparison, we also map the regression output to bucket indices to evaluate both modes under a common metric. All models are trained and evaluated within a single DBMS (e.g., PostgreSQL or SQL Server), ensuring consistency in query plan format and runtime behavior.

## 3.2 Cross-Database Adaptation Methods

### 3.2.1 Stacked Model Approach for Classic ML and QueryFormer

Cross-database cost estimation is challenging due to differences in optimizer behavior, execution engines, statistics, and runtime scale. Even identical query plans can result in very

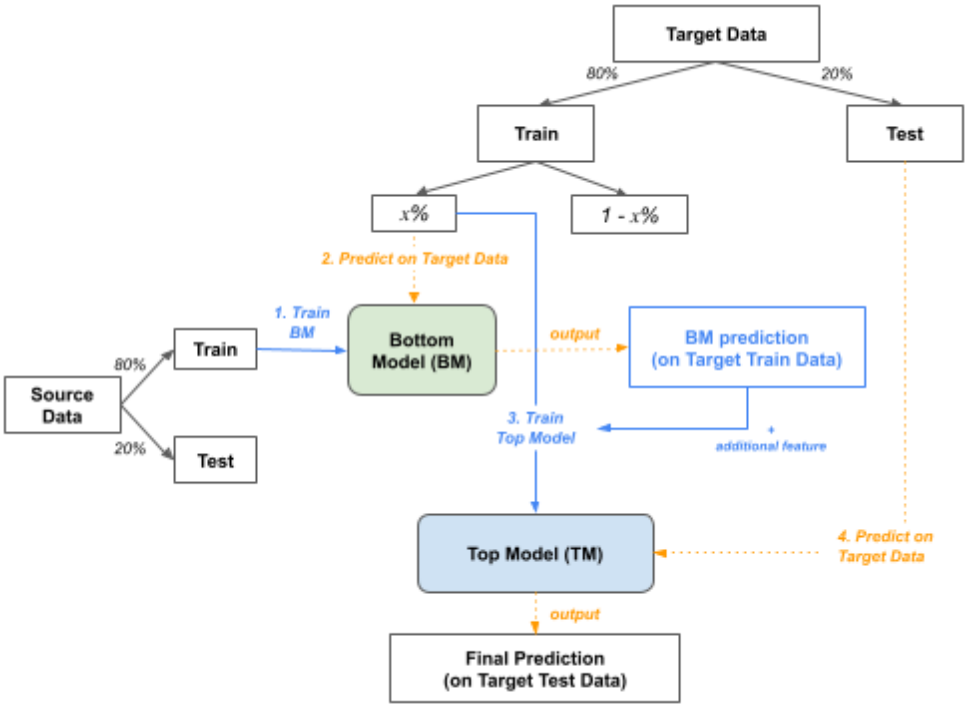


Figure 3.4: Cross-database Stacked Model Structure

different runtimes across systems due to variations in memory usage, parallelism, and caching policies.

**Stacked Model.** To address domain gaps between heterogeneous database systems, we adopt a Stacked Model approach for both Classic ML and QueryFormer (see Figure 3.4). This strategy enables knowledge transfer from a well-trained source model to a new target domain using minimal additional training data. The procedure involves four key steps:

1. **Train Bottom Model (BM):** A model is trained on a large dataset from a source database (e.g., PostgreSQL). This model captures performance patterns based on the source system's query execution behavior.

2. **Apply BM on Target Data:** The bottom model is then used to generate predictions on a small subset ($x$%, in the range of 1%–10%) of queries from the target database (e.g., MySQL). These predictions represent how the source model interprets target queries and act as cross-domain signals.

3. **Train Top Model (TM):** A top model is trained on this same target subset using two inputs: (1) Original features from the target database (e.g., MySQL `EXPLAIN` output),

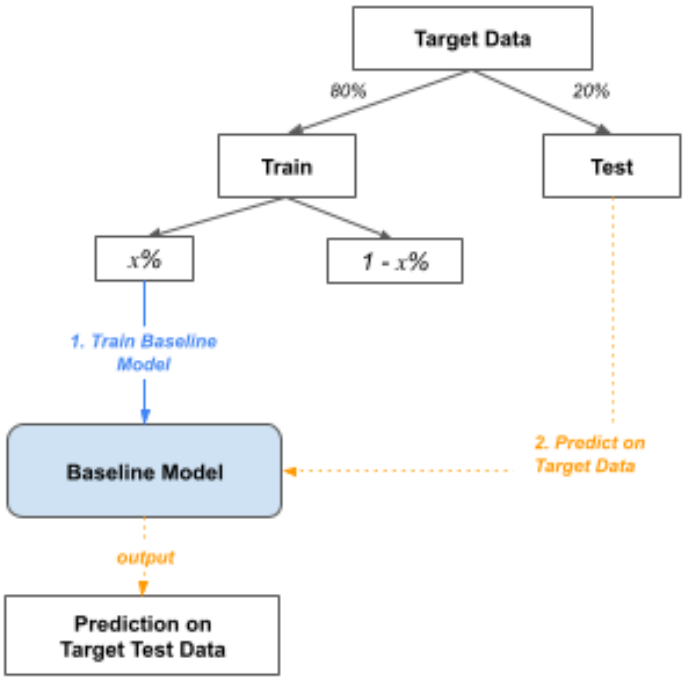


Figure 3.5: Baseline Model Structure

and (2) The predictions from the bottom model as an additional feature. This allows the top model to learn how to adjust or reinterpret the source model's outputs to better align with the target domain.

4. **Final Prediction on Target Test Set:** Once trained, the top model is used to predict runtimes on unseen target queries (i.e., the held-out target test set). This constitutes the final prediction output.

The top model effectively learns how runtime patterns from the source system relate to those of the target system, enabling improved generalization under limited target data. To evaluate transfer success, we compare this Stacked Model setup against a Baseline Model trained solely on the small target subset without cross-domain signals (see Figure 3.5). Superior performance of the Stacked Model indicates successful and efficient knowledge transfer across databases.

### 3.2.2 LLM Generalization Across Databases

Unlike tabular ML models, LLMs generalize across database systems without the need for stacked adaptation. This is because LLMs operate on high-dimensional embeddings of structured text that implicitly encode semantic and structural properties of queries. LLMs do not depend on explicit feature alignment between systems. Instead, their latent space captures patterns across execution traces, query structures, and performance distributions. As a result, an LLM fine-tuned on PostgreSQL data can be directly evaluated on MySQL queries, bypassing the need for retraining or domain-specific transformation.

This design allows us to investigate whether LLMs can serve as flexible, cross-system predictors of query performance—especially under low-data regimes where traditional methods require system-specific retraining.

## 3.3 Evaluation Metrics

To evaluate model performance, we adopt a range of metrics depending on the prediction task. These metrics are designed to capture both regression-based accuracy and classification-based interpretability in query cost estimation.

### 3.3.1 Q-error

**Q-error** is the primary metric used to evaluate regression-based models such as **Classic ML**, **QueryFormer**, and **LLM regression**, following prior work [16, 14, 13]. It measures the multiplicative deviation between the predicted cost and the actual cost, and is defined as:

$$Q\text{-}error = \max\left(\frac{P}{A}, \frac{A}{P}\right) \tag{3.1}$$

where $P$ and $A$ represent the predicted and actual execution costs, respectively. A perfect prediction yields a Q-error of 1, and lower values indicate better accuracy. This metric is particularly robust to skewed distributions and outliers, as it treats over- and under-estimation symmetrically.

### 3.3.2 Mean Relative Error (MRE)

The **Mean Relative Error (MRE)** offers a complementary, additive view of prediction quality. It quantifies the average relative deviation from ground truth, and is defined as:

$$\text{MRE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{P_i - A_i}{A_i}\right| \tag{3.2}$$

where $P_i$ and $A_i$ are the predicted and actual costs for the $i$-th query. MRE is more interpretable than Q-error, as it corresponds to the average percentage error across predictions. It is particularly useful for identifying systematic bias and is less sensitive to extreme outliers caused by small denominators.

### 3.3.3 Accuracy and Mean Bucket Difference (MBD)

For classification-based models—such as LLMs trained to predict runtime buckets—we discretize the target runtimes into 100 equally spaced buckets based on the min-max cost range. Performance is then evaluated using:

- **Accuracy:** The fraction of queries whose predicted bucket exactly matches the actual bucket.
- **Mean Bucket Difference (MBD):** The average absolute distance between the predicted and true bucket indices, defined as:

$$MBD = \frac{1}{N}\sum_{i=1}^{N}|\text{Predicted Bucket}_i - \text{Actual Bucket}_i| \tag{3.3}$$

This classification framework enables us to fairly evaluate LLMs that output class labels rather than continuous values, and also provides an interpretable way to compare regression and classification approaches.

### 3.3.4 Cross-task Metric Estimation

To ensure fair comparisons between regression and classification models:

- **For classification models:** Regression-specific metrics (Q-error and MRE) are estimated by assigning each predicted bucket its midpoint cost and comparing this to the actual runtime (labeled as Est.).
- **For regression models:** Classification-specific metrics (Accuracy and MBD) are estimated by mapping predicted continuous costs to their corresponding bucket index (labeled as Est.).

This unified evaluation strategy ensures consistent metric reporting across heterogeneous model architectures and reveals trade-offs between numeric precision and categorical classification.

# CHAPTER 4

# Experiment

## 4.1 Training Infrastructure and Performance

**Training Environment.** In the Classic ML method, models were trained locally on CPU (Intel Core i7-10875H, 8 cores, 16 threads). QueryFormer and LLM-based methods were trained on GPU resources from the MIG40 and SPGPU partitions from University of Michigan's HPC cluster. MIG40 provides access to NVIDIA A100 80GB GPUs with up to 156 TFLOPS peak performance, while SPGPU features NVIDIA A40 GPUs with 48GB memory and 74.8 TFLOPS performance. GPU partition selection depended on availability.

**Training and Inference Costs.** Table 4.1 compares cross-database training and inference costs for Classic ML, QueryFormer, and LLM (CLA) methods in query cost prediction. Training costs reflect the time required to train on 80% of 2,200 queries (i.e., 1,760 queries), while inference costs are reported per query. Classic ML, executed on CPUs, incurs moderate training times—417.56s for PG to MS and 1397.91s for MS to PG—but exhibits significantly higher inference latency, particularly for PG to MS (9.18s/query). Despite its simplicity, this makes Classic ML less suitable for latency-sensitive or real-time applications.

In contrast, QueryFormer demonstrates both efficient training (under 300s) and extremely low inference cost (0.00059s/query), leveraging GPU acceleration for rapid deployment and scalability. LLM Classification, while offering potentially higher modeling capacity, requires substantially more training time (up to 10,108s) and exhibits moderately low inference latency (0.74–2.42s/query). Overall, QueryFormer provides the most favorable trade-off between training cost and inference efficiency, making it a strong candidate for latency-sensitive workloads, whereas LLMs may be better suited for tasks emphasizing model performance over runtime constraints. Note that MS to PG takes longer to train than PG to MS for Classic ML and LLM due to the increased complexity of the raw query plan format in MS.

This is particularly evident in the LLM-based method, where the tokenized input for MS plans is approximately three times longer than that of PG plans.

Table 4.1: Training and test costs (s)

| Method | Device | Databases | Bottom | Top | Train Total | Test per query |
|---|---|---|---|---|---|---|
| Classic ML | CPU | PG to MS | 400.97 | 16.59 | 417.56 | 9.18 |
| Classic ML | CPU | MS to PG | 1329.17 | 68.74 | 1397.91 | 1.11 |
| QF | GPU | PG to MS | 198.70 | 86.61 | 285.31 | 0.00059 |
| QF | GPU | MS to PG | 195.68 | 63.70 | 259.38 | 0.00059 |
| LLM (CLA) | GPU | PG to MS | 1302.00 | 529.00 | 1331.00 | 2.42 |
| LLM (CLA) | GPU | MS to PG | 9945.00 | 163.00 | 10108.00 | 0.74 |

*Note:* "Bottom" refers to models trained only on source database data. "Top" refers to stacked models that take predictions from the Bottom model (on $x\%$ target data) as additional input features, and are then trained on $x\%$ of target database data.

## 4.2 Same-database Cost Estimation

**Experiment Setup.** In this section, we evaluate the performance of regression-based methods and LLM-based classification models in the cost estimation task within the database.

**Metrics.** For regression methods, we report metrics that include the Q error, the mean relative error (MRE), the estimated accuracy and the estimated mean bucket distance (MBD). For LLM classification models, we report the estimated Q error, the estimated MRE, the accuracy, and the MBD, consistent with the definitions outlined in Section 3.3. All QueryFormer models are trained 200 epochs, and all LLaMA models are finetuned 30 epochs.

**Observations.** From Table 4.2, the **LLaMA 3-8B CLA** model delivers the highest overall performance, achieving top accuracy on both MS (88.41%) and PG (75.45%), along with the lowest mean bias distance (0.17 and 0.46, respectively). This indicates strong predictive precision and stability. **Classic ML** performs best on low-quantile metrics ($q_{50}$ of 1.02 on MS, 1.03 on PG; MRE of 0.05 on MS), making it reliable for median-range estimates but less robust under long-tail queries, particularly on PG. **QueryFormer** excels in tail stability with the lowest $q_{99}$ and $q_{\max}$ on PG (1.19 and 1.32), but its overall accuracy and bias metrics lag behind LLaMA-based models.

**Selection of LLM-Based Method.** When comparing the regression (REG) and classification (CLA) variants of LLaMA, the CLA models significantly outperform their REG counterparts in both accuracy and stability. For instance, LLaMA 3-8B CLA achieves 88.41%

Table 4.2: Same-DB Cost Estimation Performance

| Model | DB | $q_{50}$ | $q_{90}$ | $q_{99}$ | $q_{\max}$ | MRE | Accuracy (%) | Avg Dist |
|---|---|---|---|---|---|---|---|---|
| AutoGluon | MS | 1.02 | 1.13 | 1.22 | 1.34 | 0.05 | 76.82 | 0.41 |
| QueryFormer | MS | 1.08 | 1.12 | 1.21 | 1.60 | 0.08 | 41.82 | 1.09 |
| LLaMA 3-8B REG | MS | 1.28 | 1.77 | 2.75 | 8.59 | 0.29 | 25.68 | 4.48 |
| LLaMA 3-3B REG | MS | 1.11 | 1.45 | 1.76 | 1.87 | 0.14 | 39.70 | 3.08 |
| LLaMA 3-8B CLA | MS | 1.10 | 2.80 | 3.17 | 3.23 | 0.24 | 88.41 | 0.17 |
| LLaMA 3-3B CLA | MS | 1.10 | 2.80 | 3.17 | 3.23 | 0.24 | 87.27 | 0.18 |
| AutoGluon | PG | 1.03 | 2.14 | 2.52 | 2.67 | 0.15 | 46.14 | 3.68 |
| QueryFormer | PG | 1.05 | 1.12 | 1.19 | 1.32 | 0.06 | 37.58 | 1.65 |
| LLaMA 3-8B REG | PG | 1.24 | 1.68 | 6.06 | 12.76 | 0.31 | 14.09 | 5.41 |
| LLaMA 3-3B REG | PG | 1.16 | 1.28 | 1.38 | 4.30 | 0.17 | 10.91 | 3.66 |
| LLaMA 3-8B CLA | PG | 1.10 | 1.72 | 2.86 | 2.88 | 0.14 | 75.45 | 0.46 |
| LLaMA 3-3B CLA | PG | 1.12 | 1.72 | 2.86 | 2.88 | 0.15 | 73.86 | 0.52 |

accuracy on MS compared to only 25.68% by the REG version. Similarly, on PG, CLA improves accuracy by over 60 percentage points while reducing MBD by a factor of ten.

Based on this evaluation, we select LLaMA 3-8B CLA as the final large language model for cost estimation tasks. It not only outperforms other models across the majority of evaluation metrics, but also offers a strong balance between accuracy, robustness, and low bias. Its classification-based design mitigates extreme prediction errors, making it particularly suitable for integration into cost-sensitive applications such as query optimization and resource planning in DBMSs.

## 4.3 Cross-database Cost Estimation

**Experiment Setup.** In this section, we evaluate the transferability of our models across heterogeneous database systems. Our focus is on transferring between PostgreSQL and MySQL, though future work should consider a broader range of database systems to evaluate generalizability. For all three methods, we begin by training a **Baseline Model** using only a small fraction of data from the target database (see Figure 3.5 for the model setup). For both Classic ML and QueryFormer (QF), we then train a **Stacked Model** (illustrated in Figure 3.4), which integrates learned knowledge from the source database along with a limited amount of target database data. In contrast, as detailed in Section 3.2.2, the LLM-based method does not require stacking. Instead, we directly continue fine-tuning the model initially trained on the source database using target database data.

**Metrics.** We use the same metrics as the same-database cost estimation. All QueryFormer bottom and top models are trained 200 epochs. All LLaMA models are first finetuned 30 epochs on source database, and then finetuned for 100 epochs on target database.

**Result Organization.** The results are organized into six groups based on the proportion of data from the target database used for training (1%, 3%, 5%, 7%, 9%, 10%). Within each table corresponding to a fixed proportion, we first report the performance of the baseline method across all three modeling approaches, followed by the performance of the stacked method. For each data proportion, we present results from PostgreSQL to MySQL transfer first, followed by MySQL to PostgreSQL.

**LLM Model Selection.** For LLM-based methods, we chose to focus on the LLaMA 3–8B classification model due to its superior performance compared to other variants of LLM, as demonstrated in Section 4.2.

### 4.3.1 1% Target Database Data

Table 4.3: Cross-DB Prediction (PostgreSQL to MySQL, with 1% MySQL Data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.29 | 3.28 | inf | inf | 0.54 | 29.55 | 4.91 |
| QF | B | 1.60 | 10.47 | 41.76 | 44.07 | 0.51 | 23.94 | 8.97 |
| LLM | B | 1.35 | 11.99 | 34.72 | 37.95 | 2.65 | 49.55 | 11.55 |
| Classic ML | S | 1.27 | 3.49 | inf | inf | 0.53 | 25.00 | 4.94 |
| QF | S | 1.43 | 7.15 | 39.06 | 41.17 | 0.65 | 25.68 | 7.44 |
| LLM | S | 1.35 | 6.73 | 34.58 | 37.95 | 2.40 | 54.09 | 10.39 |

Note: Q-errors reports inf when prediction is very close to zero.

**Table 4.3 Observations (PostgreSQL to MySQL).** In this direction, comparing across methods, although LLM achieves the highest Estimated Accuracy (49.55% for B and 54.09% for S), it also suffers from the worst MRE (2.65 for B and 2.40 for S) and largest Est. MBD, indicating highly dispersed and less reliable predictions. Classic ML performs best on low quantiles such as $q_{50}$ and $q_{90}$, but its $q_{99}$ and $q_{\max}$ are inf, likely due to close-to-zero underpredictions inflating ratios. Comparing Baseline vs. Stacked, QF demonstrates the most notable improvement with a substantial reduction in $q_{90}$ and MRE and an increase in Estimated Accuracy from 23.94% to 25.68%. In contrast, stacking leads to only marginal improvement for LLM and slightly reduces performance for Classic ML.

**Table 4.4 Observations (MySQL to PostgreSQL).** In this reverse direction, LLM again delivers the highest Estimated Accuracy (34.55% for B and 34.77% for S), but at the

Table 4.4: Cross-DB Prediction (MySQL to PostgreSQL, with 1% of PostgreSQL Data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.49 | 3.66 | 4.45 | 4.56 | 0.40 | 1.14 | 12.62 |
| QF | B | 1.31 | 6.98 | 19.10 | 20.45 | 1.01 | 24.02 | 7.87 |
| LLM | B | 1.16 | 3.87 | 31.27 | 31.45 | 1.92 | 34.55 | 10.06 |
| Classic ML | S | 3.48 | 7.18 | 10.36 | 11.13 | 2.81 | 5.45 | 29.40 |
| QF | S | 1.26 | 7.75 | 34.58 | 35.55 | 0.35 | 29.09 | 7.03 |
| LLM | S | 1.16 | 4.98 | 10.90 | 11.54 | 1.28 | 34.77 | 9.47 |

cost of high MRE (1.92 for B and 1.28 for S) and Est. MBD. Classic ML-B achieves the best $q_{50}$ and $q_{90}$ values but struggles with high variability in $q_{99}$ and $q_{\max}$ in the stacked version, where performance degrades significantly (MRE = 2.81, Est. Accuracy = 5.45%). QF, however, benefits consistently from stacking, showing a drop in MRE from 1.01 to 0.35 and a boost in Estimated Accuracy from 24.02% to 29.09%, suggesting improved robustness.

**Main Takeaways (1%).** *LLM consistently yields the highest accuracy across Baseline and Stacked model, while MRE of Classic ML and QF outperform that of LLM.* ***Cross-Database Knowledge Transfer*** *While Stacked Model improved LLM performance by a moderate amount (particularly in reducing high q-error quantiles for MS-to-PG transfer), it leads to unstable and less consistent improvements for Classic ML and QF.*

### 4.3.2 3% Target Database Data

Table 4.5: Cross-DB Prediction (PostgreSQL to MySQL, with 3% of MySQL Data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.11 | 1.67 | 2.07 | 2.27 | 0.25 | 36.93 | 2.49 |
| QF | B | 1.17 | 1.83 | 10.21 | 11.13 | 0.24 | 35.23 | 3.32 |
| LLM | B | 1.22 | 3.18 | 7.80 | 8.06 | 0.47 | 69.55 | 1.42 |
| Classic ML | S | 1.08 | 1.56 | 4.23 | 7.57 | 0.19 | 40.45 | 2.08 |
| QF | S | 1.28 | 3.43 | 15.88 | 18.82 | 0.51 | 29.47 | 5.02 |
| LLM | S | 1.22 | 2.81 | 3.23 | 9.57 | 0.39 | 71.82 | 1.06 |

**Table 4.5 Observations (PostgreSQL to MySQL)**. At the 3% level, LLM maintains its lead in Estimated Accuracy (69.55% for B and 71.82% for S) and now achieves the lowest Est. MBD, suggesting more reliable average predictions. However, its high $q_{99}$ and $q_{\max}$ values indicate lingering tail issues. Classic ML continues to excel on $q_{50}$ and $q_{90}$ with

modest stacking gains. In contrast, QF's performance deteriorates, especially in the stacked setting, with pronounced spikes in upper quantiles and overall variability.

Table 4.6: Cross-DB Prediction (MySQL to PostgreSQL, with 3% of PostgreSQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.16 | 2.39 | 2.56 | 3.10 | 0.22 | 21.82 | 8.48 |
| QF | B | 1.08 | 1.18 | 1.97 | 2.53 | 0.09 | 47.57 | 1.25 |
| LLM | B | 1.15 | 1.80 | 7.48 | 7.52 | 0.53 | 60.23 | 1.80 |
| Classic ML | S | 2.19 | 6.63 | 9.37 | 9.43 | 2.10 | 10.91 | 25.78 |
| QF | S | 1.16 | 3.06 | 15.93 | 16.43 | 0.48 | 33.41 | 5.08 |
| LLM | S | 1.15 | 1.80 | 3.32 | 3.34 | 0.27 | 60.23 | 1.31 |

**Table 4.6 Observations (MySQL to PostgreSQL)**. In the reverse direction, LLM sustains the highest accuracy (60.23% for both B and S) and remains strong in MBD. Classic ML performs reliably on lower quantiles, but its stacked variant shows major instability (MRE = 2.10, MBD = 25.78). QF again demonstrates strong baseline precision (MRE = 0.09) but suffers from substantial error inflation in the stacked model.

**Main Takeaways (3%).** *LLM maintains its lead in accuracy and now shows improved MBD, while Classic ML dominates low q-error metrics and both Classic ML and QF obtain better MRE across Baseline and Stacked model.* ***Cross-Database Knowledge Transfer:*** *While stacking provides moderate gains for LLMs, it leads to unstable and less consistent improvements for Classic ML and QF, consistent with the findings at 1%*

### 4.3.3 5% Target Database Data

Table 4.7: Cross-DB Prediction (PostgreSQL to MySQL, with 5% of MySQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.03 | 1.13 | 1.40 | 1.49 | 0.05 | 68.18 | 0.54 |
| QF | B | 1.23 | 2.30 | 23.59 | 25.06 | 0.33 | 26.89 | 5.04 |
| LLM | B | 1.10 | 2.81 | 3.22 | 3.24 | 0.33 | 80.00 | 0.44 |
| Classic ML | S | 1.07 | 1.42 | 1.54 | 3.00 | 0.10 | 38.30 | 0.95 |
| QF | S | 1.20 | 1.76 | 17.37 | 18.94 | 0.27 | 32.35 | 3.26 |
| LLM | S | 1.16 | 2.81 | 3.22 | 3.24 | 0.34 | 75.68 | 0.47 |

**Table 4.7 Observations (PostgreSQL to MySQL)**. LLM continues its trend of high Estimated Accuracy (80.00% for B) and lowest MBD, reinforcing its reliability across splits.

Classic ML remains strong on lower quantiles but shows increased tail errors in the stacked version. Notably, QF—despite severe tail errors in its Baseline model ($q_{\max} = 25.06$)—sees modest improvement from stacking, with reduced MRE and MBD.

Table 4.8: Cross-DB Prediction (MySQL to PostgreSQL, with 5% of PostgreSQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.14 | 2.21 | 2.67 | 3.29 | 0.21 | 28.18 | 5.25 |
| QF | B | 1.06 | 1.31 | 24.81 | 26.53 | 0.86 | 31.65 | 1.94 |
| LLM | B | 1.11 | 1.80 | 7.48 | 7.52 | 0.51 | 66.59 | 1.42 |
| Classic ML | S | 3.01 | 6.24 | 8.22 | 13.71 | 2.18 | 10.45 | 25.94 |
| QF | S | 1.37 | 3.09 | 3.44 | 3.57 | 0.43 | 37.27 | 4.10 |
| LLM | S | 1.11 | 1.80 | 7.48 | 7.52 | 0.51 | 66.82 | 1.41 |

**Table 4.8 Observations (MySQL to PostgreSQL)**. LLM again demonstrates stable performance, while Classic ML's stacked variant continues to degrade sharply in tail metrics. A key shift here is QF's clear improvement from stacking: dramatic reductions in $q_{99}$ and $q_{\max}$ and a sizable boost in Estimated Accuracy to 37.27%, indicating more robust generalization in this direction at 5

**Main Takeaways (5%).** *Consistent to findings from previous proportions, LLM maintains its lead in accuracy and now shows improved MBD, while Classic ML dominates low q-error metrics and both Classic ML and QF obtain better MRE across Baseline and Stacked model.* ***Cross-Database Knowledge Transfer:*** *At this proportion, stacking yields stable gains for QF—whereas it had previously offered more consistent improvements for LLM—but shows less reliable benefits for both Classic ML and LLM.*

### 4.3.4 7% Target Database Data

Table 4.9: Cross-DB Prediction (PostgreSQL to MySQL, with 7% of MySQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.02 | 1.12 | 1.32 | 1.96 | 0.04 | 74.20 | 0.48 |
| QF | B | 1.19 | 2.19 | 21.68 | 23.03 | 0.29 | 31.06 | 4.92 |
| LLM | B | 1.15 | 2.80 | 3.17 | 3.23 | 0.24 | 82.73 | 0.23 |
| Classic ML | S | 1.04 | 1.17 | 1.35 | 2.20 | 0.07 | 73.98 | 0.49 |
| QF | S | 1.19 | 1.74 | 10.56 | 11.89 | 0.26 | 37.05 | 3.20 |
| LLM | S | 1.16 | 2.80 | 3.17 | 3.23 | 0.24 | 79.77 | 0.26 |

**Table 4.9 Observations (PostgreSQL to MySQL).** LLM maintains its lead with top accuracy (82.73% for B) and minimal MBD, showing excellent stability. Classic ML continues to perform well on lower quantiles with only slight stacking degradation. QF, while still suffering from extreme tail errors in its Baseline model, shows moderate improvement post-stacking—reducing $q_{99}$ and boosting accuracy to 37.05

Table 4.10: Cross-DB Prediction (MySQL to PostgreSQL, with 7% of PostgreSQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.06 | 2.09 | 2.56 | 4.16 | 0.14 | 32.95 | 3.94 |
| QF | B | 1.06 | 1.30 | 2.20 | 2.26 | 0.09 | 57.88 | 1.49 |
| LLM | B | 1.12 | 1.72 | 2.86 | 2.88 | 0.16 | 72.05 | 0.79 |
| Classic ML | S | 2.77 | 7.74 | 9.29 | 10.09 | 2.07 | 0.23 | 25.95 |
| QF | S | 1.09 | 1.33 | 1.54 | 1.70 | 0.12 | 46.14 | 1.46 |
| LLM | S | 1.12 | 1.72 | 2.86 | 2.88 | 0.16 | 72.27 | 0.79 |

**Table 4.10 Observations (MySQL to PostgreSQL).** LLM's accuracy and stability persist. Classic ML, however, collapses entirely when stacked—accuracy plummets to 0.23%, and tail metrics spike drastically. QF demonstrates its most balanced performance yet, significantly reducing MRE and tail errors while retaining strong accuracy, continuing its trend of stacking gains.

**Main Takeaways (7%).** *Consistent with previous proportions, the LLM approach continues to lead in both accuracy and Mean Bucket Distance (MBD). However, now both Classic ML and QF now outperform LLM on low q-error metrics (but similar MRE) across both Baseline and Stacked models.* ***Cross-Database Knowledge Transfer:*** *Similar to 5%, stacking yields stable gains for QF, but shows less reliable benefits for both Classic ML and LLM. Notably, at this higher proportion, stacking significantly degrades performance for the MS-to-PG transfer when using the Classic ML method.*

### 4.3.5 9% Target Database Data

**Table 4.11 Observations (PostgreSQL to MySQL).** LLM sustains peak performance, achieving 86.59% accuracy and the lowest MBD, with consistent behavior across quantiles. Classic ML, though strong in the Baseline model, deteriorates sharply when stacked—most notably in accuracy (dropping by over 50 percentage points). QF remains steady but struggles with extreme outliers, showing only modest gains from stacking.

**Table 4.12 Observations (MySQL to PostgreSQL).** LLM continues to lead with high accuracy and stable metrics, unaffected by stacking. Classic ML collapses in the stacked

Table 4.11: Cross-DB Prediction (PostgreSQL to MySQL, with 9% of MySQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.02 | 1.10 | 1.36 | 1.47 | 0.04 | 73.30 | 0.46 |
| QF | B | 1.10 | 1.47 | 3.89 | 5.24 | 0.19 | 41.52 | 2.51 |
| LLM | B | 1.10 | 2.80 | 3.17 | 3.23 | 0.24 | 86.59 | 0.19 |
| Classic ML | S | 1.19 | 1.43 | 2.75 | 2.82 | 0.27 | 19.77 | 2.65 |
| QF | S | 1.17 | 2.01 | 3.53 | 5.38 | 0.34 | 34.62 | 2.90 |
| LLM | S | 1.10 | 2.80 | 3.17 | 3.23 | 0.24 | 85.23 | 0.21 |

Table 4.12: Cross-DB Prediction (MySQL to PostgreSQL, with 9% of PostgreSQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.04 | 2.10 | 2.32 | 4.06 | 0.13 | 41.59 | 3.75 |
| QF | B | 1.11 | 1.49 | 4.36 | 4.72 | 0.23 | 44.17 | 2.74 |
| LLM | B | 1.11 | 1.72 | 2.68 | 2.88 | 0.15 | 73.64 | 0.53 |
| Classic ML | S | 2.97 | 7.46 | 11.11 | 11.97 | 2.55 | 0.00 | 27.58 |
| QF | S | 1.08 | 1.24 | 1.33 | 1.45 | 0.09 | 58.79 | 1.28 |
| LLM | S | 1.11 | 1.72 | 2.86 | 2.88 | 0.15 | 73.41 | 0.54 |

mode with near-zero accuracy and spiked tail errors. QF, by contrast, shows its strongest stacking benefit yet—substantially lowering tail risks and achieving nearly 59% accuracy.

**Main Takeaways (9%).** *The LLM approach continues to lead in both accuracy and Mean Bucket Distance (MBD), though Classic ML shows signs of closing the gap. As observed at the 7% level, both Classic ML and QF outperform LLM on low q-error metrics across both Baseline and Stacked models.* ***Cross-Database Knowledge Transfer:*** *Consistent with the 5% and 7% setting, stacking yields stable improvements for QF. However, similar to findings from 7%, stacking does not enhance the performance of the LLM method and even degrades the performance of Classic ML.*

### 4.3.6 10% Target Database Data

**Table 4.13 Observations (PostgreSQL to MySQL).** LLM continues to excel with top accuracy (up to 86.82%) and the lowest MBD, reinforcing its stability with more target data. Classic ML's strong baseline performance again breaks down when stacked, showing significant drops in accuracy and increases in error. QF benefits modestly from stacking, improving across all metrics but still trailing LLM.

**Table 4.14 Observations (MySQL to PostgreSQL).** LLM remains highly consis-

Table 4.13: Cross-DB Prediction (PostgreSQL to MySQL, with 10% of MySQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.03 | 1.14 | 1.27 | 1.46 | 0.05 | 71.25 | 0.56 |
| QF | B | 1.15 | 1.61 | 3.32 | 3.52 | 0.20 | 40.23 | 3.65 |
| LLM | B | 1.15 | 2.80 | 3.17 | 3.23 | 0.24 | 84.32 | 0.22 |
| Classic ML | S | 1.06 | 1.41 | 1.64 | 1.76 | 0.13 | 51.25 | 0.76 |
| QF | S | 1.11 | 1.36 | 2.00 | 3.38 | 0.16 | 43.71 | 1.50 |
| LLM | S | 1.15 | 2.80 | 3.17 | 3.23 | 0.24 | 86.82 | 0.18 |

Table 4.14: Cross-DB Prediction (MySQL to PostgreSQL, with 10% of PostgreSQL data)

| Method | Model | q_50 | q_90 | q_99 | q_max | MRE | Accuracy | MBD |
|---|---|---|---|---|---|---|---|---|
| Classic ML | B | 1.06 | 2.16 | 2.56 | 3.95 | 0.14 | 37.27 | 3.83 |
| QF | B | 1.11 | 1.26 | 1.42 | 1.44 | 0.12 | 43.04 | 2.12 |
| LLM | B | 1.13 | 1.72 | 2.86 | 2.88 | 0.16 | 72.95 | 0.58 |
| Classic ML | S | 2.79 | 6.65 | 8.29 | 8.41 | 2.22 | 0.00 | 26.99 |
| QF | S | 1.06 | 1.15 | 1.23 | 1.30 | 0.07 | 57.43 | 0.88 |
| LLM | S | 1.13 | 1.72 | 2.86 | 2.88 | 0.16 | 71.59 | 0.60 |

tent with strong accuracy and low dispersion. QF shows its best stacked performance to date—achieving minimal MRE and the lowest tail errors across all models. Classic ML's stacked version collapses once more, with 0% accuracy and a surge in MBD.

**Main Takeaways (10%).** *As noted previously, the LLM approach continues to lead in both accuracy and Mean Bucket Distance (MBD), with Classic ML steadily narrowing the gap. Both Classic ML and QueryFormer (QF) outperform LLM on low q-error metrics across both Baseline and Stacked models.* ***Cross-Database Knowledge Transfer:*** *Consistent with the 5%, 7%, and 9% settings, stacking provides stable improvements for QF. However, similar to the 7% and 9% case, stacking fails to improve the performance of the LLM method and even degrades the performance of Classic ML.*

# CHAPTER 5

# Conclusion

**1. Cross-Database Knowledge Transfer.** The effectiveness of stacking varies notably across methods:

**Classic ML.** Classic ML models (e.g., XGBoost, Random Forest) rely heavily on the statistical distributions of input features. However, different databases often produce divergent query plan structures, cost estimations, and operator usage patterns. Since Classic ML constructs query run times by predicting at the operator level and summing the results, it struggles to generalize across these domain shifts. Additionally, Classic ML depends on fixed, manually engineered features, which limits its ability to capture the structural complexity of query plans—unlike QueryFormer's tree-aware architecture or LLMs' token-level semantic modeling—resulting in weaker cross-database transferability.

**QueryFormer.** QF demonstrates strong cross-database transfer performance, as it is explicitly designed to encode tree-structured query plans using Transformer-based architectures. This enables it to capture both local operator characteristics and global structural context, enhancing robustness across DBMSs. Stacking consistently improves QF performance, especially at higher proportions of target database data. This suggests that a sufficient quantity of target data is needed to mitigate source bias and support meaningful knowledge transfer.

**LLM Finetune.** Despite achieving the highest accuracies and lowest mean bucket distances, LLMs show limited improvement from stacking. This is likely due to their pretraining on large, diverse corpora, which provides strong general-purpose representations. When fine-tuned on even a small fraction of target database data, LLMs can quickly adapt to system-specific patterns, reducing the added value of transferring knowledge from the source. As a result, the impact of stacking becomes negligible, with most performance gains coming directly from target-domain fine-tuning.

**2. Model Performance Across Metrics.** While transferability varies across methods, the overall performance trends also differ significantly depending on the evaluation metric.

The LLM-based approach consistently achieves the highest accuracy and lowest Mean Bucket Distance (MBD) across both baseline and stacked settings. This reflects the model's strong global understanding of query semantics and cost patterns, likely enabled by its pre-training on large, diverse corpora and adaptability during fine-tuning. In contrast, Classic ML performs particularly well on low $q$-error quantiles (e.g., $q_{50}$), especially at higher proportions of target data. This strength stems from its operator-level granularity, which allows it to make precise local predictions. However, it often struggles with higher quantiles and overall accuracy due to its limited global view and sensitivity to distribution shifts between source and target databases. QueryFormer (QF) strikes a balance across metrics. While it may not match the LLM in raw accuracy or MBD, it shows consistent performance across all $q$-error metrics, including both low and high quantiles. This stability is attributed to its tree-aware design, which effectively captures both local and global aspects of query plans. Furthermore, its performance is notably enhanced by stacking, which allows it to integrate transferable knowledge from the source while adapting to the target database.

**3. Trade-offs in Efficiency versus Performance** Based on empirical training and inference cost data (Table 4.1), QueryFormer (QF) emerges as the most efficient method overall, achieving the fastest training times (259.38–285.31 seconds) and near-instantaneous inference latency (0.00059s per query) on GPU. In contrast, LLM (CLA) requires substantially more training time, especially for the MS to PG transfer task (over 10,000 seconds), and shows moderate inference latency (0.74–2.42s), making it the most computationally demanding model. Classic ML, trained on CPU, falls between these two extremes—its training times range from 417.56s to 1397.91s, and inference takes 1.11–9.18s per query depending on the transfer direction. While Classic ML could benefit from GPU acceleration in practice, its reliance on traditional ML pipelines makes it less optimized for modern hardware parallelism. Overall, QF achieves the best trade-off between speed and predictive improvement when stacked, LLM excels in accuracy but at high resource cost, and Classic ML remains a viable baseline with moderate efficiency and simplicity.

**4. Implementation Complexity and Generalization.** From an engineering standpoint, Classic ML and LLM are relatively straightforward to implement and adapt to new databases or benchmarks, largely due to their reliance on general tabular data structures and input features. QF, on the other hand, poses significantly more implementation challenges—it requires database-specific query plan parsing, operator embedding alignment, and careful design of attention architectures suited for varying schemas. This complexity can hinder quick experimentation or transfer to new DBMS environments without tailored ad-

justments.

**5. Overall Recommendations and Future Directions.** Our results suggest that the choice of method and transfer strategy should be guided by both the evaluation goals (e.g., accuracy vs. low $q$-error performance) and the availability of target database data. LLM fine-tuning is a strong default choice when even a small amount of target data is available, as it provides high accuracy and generalization with minimal need for additional transfer mechanisms like stacking. However, due to diminishing returns from knowledge transfer, its performance gains plateau quickly with increased stacking complexity. In contrast, QueryFormer is a compelling option when both source and target databases are available and moderate target data proportions can be used for transfer. Its structure-aware design and responsiveness to stacking make it well-suited for robust performance across a range of metrics, including consistent low and high $q$-error bounds. Classic ML, while less flexible and more sensitive to domain shifts, still offers strong operator-level precision and benefits from stacking when sufficient target data is present—particularly in tasks where low $q$-error quantiles are critical.

For future work, we recommend investigating hybrid models that combine the structural awareness of QueryFormer with the language-driven generalization capabilities of LLMs. Additionally, incorporating unsupervised or semi-supervised pretraining directly on target DBMS logs could enhance adaptability without requiring labeled data. Finally, better understanding when and how to apply stacking—especially for large models—remains an open direction for improving cross-database transfer in low-resource scenarios.

# BIBLIOGRAPHY


[1] Gao Cong, Jingyi Yang, and Yue Zhao. Machine learning for databases: Foundations, paradigms, and open problems. pages 622–629, 06 2024.

[2] Aaron Grattafiori et al. The llama 3 herd of models, 2024.

[3] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), March 2023.

[4] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *CoRR*, abs/1809.00677, 2018.

[5] Weirui Kuang, Bingchen Qian, Zitao Li, Daoyuan Chen, Dawei Gao, Xuchen Pan, Yuexiang Xie, Yaliang Li, Bolin Ding, and Jingren Zhou. Federatedscope-llm: A comprehensive package for fine-tuning large language models in federated learning, 2023.

[6] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. Gptuner: A manual-reading database tuning system via gpt-guided bayesian optimization. *Proceedings of the VLDB Endowment*, 17(8):1939–1952, April 2024.

[7] Wan Shen Lim, Lin Ma, William Zhang, Matthew Butrovich, Samuel Arch, and Andrew Pavlo. Hit the gym: Accelerating query execution to efficiently bootstrap behavior models for self-driving database management systems. *Proc. VLDB Endow.*, 17(11):3680–3693, July 2024.

[8] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 1275–1288, New York, NY, USA, 2021. Association for Computing Machinery.

[9] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *CoRR*, abs/1904.03711, 2019.

[10] Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial*

*Intelligence Techniques for Data Management*, aiDM'18, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Shvetank Prakash and Vijay Janapa Reddi. The QuArch Experiment: Crowdsourcing a Machine Learning Dataset for Computer Architecture, 2024.

[12] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[13] Ji Sun and Guoliang Li. An end-to-end learning-based cost estimator. *Proc. VLDB Endow.*, 13(3):307–319, November 2019.

[14] Liane Vogel, Benjamin Hilprecht, and Carsten Binnig. Towards foundation models for relational databases [vision paper], 2023.

[15] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. Queryformer: a tree transformer model for query plan representation. *Proc. VLDB Endow.*, 15(8):1658–1670, April 2022.

[16] Yue Zhao, Zhaodonghui Li, and Gao Cong. A comparative study and component analysis of query plan representation techniques in ml4db studies. *Proc. VLDB Endow.*, 17(4):823–835, December 2023.