TATA: An Efficient Framework for Task Transfer in Query Plan Representation

Yue Zhao
Nanyang Technological University
Singapore
zhao0342@e.ntu.edu.sg

Songsong Mo*
Nanyang Technological University
Singapore
songsong.mo@ntu.edu.sg

Gao Cong*
Nanyang Technological University
Singapore
gaocong@ntu.edu.sg

ABSTRACT

Machine learning for database systems has achieved significant success in various database components, such as cost estimation, query optimization, index selection, view recommendation, and semantic equivalence detection. However, these solutions typically focus on a single task and normally need a large amount of labeled data for the task to train machine learning models. Even if a solution can be adapted for a different task, it will require recollecting labeled data for each new task, which is typically much more time-consuming than model training. While dataset collection is relatively easier for some tasks, it can be prohibitively expensive for others. A natural solution is to use transfer learning techniques to adapt learned knowledge from one task to another. However, we show that naive transfer learning methods perform poorly and are only as good as training from scratch. Their failures are mainly due to three challenges: (1) the source model is not robust as it is optimized to its task only; (2) the size of the target dataset is small; and (3) the inevitable distribution shift when changing tasks.

To overcome these challenges, we first study the task transfer problem in query plan representation and propose a new framework TATA for the problem. Specifically, to address the lack of robustness in the source model, TATA incorporates a self-supervised component during the pretraining stage. Specifically, we design a query plan decoder to reconstruct the original query plan from its representation, ensuring the model preserves key features. This leads to more robust and transferable query plan representations. Next, to address the issues of small datasets and distribution shift, TATA generates an arbitrary number of query plans for the target task and assigns them realistic pseudo labels. This is achieved by utilizing both strong database domain knowledge and available datasets. Through extensive experiments, we show that TATA delivers substantial improvements on task transfer, achieving up to 5× reduction in dataset collection cost when transferring from cost estimation to two representative target tasks: query optimization and index selection. We demonstrate compatibility with three distinct query plan representation models, establishing broader applicability than prior transfer approaches.

PVLDB Reference Format:
Yue Zhao, Songsong Mo, and Gao Cong. TATA: An Efficient Framework for Task Transfer in Query Plan Representation. PVLDB, 19(3): 413 - 425, 2025.
doi:10.14778/3778092.3778102

*contact authors.

Proceedings of the VLDB Endowment, Vol. 19, No. 3 ISSN 2150-8097.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/zhaoyue-ntu/tata.

1 INTRODUCTION

Machine learning for databases (ML4DB) systems have made tremendous advances in recent years. Among them, query-plan-driven systems have emerged as a particularly successful direction. These systems take physical query execution plans as input and address a wide range of core database tasks, such as cost estimation [11, 20, 30], index selection [4, 29], and query optimization [17, 18, 38]. By leveraging powerful machine learning techniques to accurately capture query execution patterns and workload characteristics, these systems can significantly outperform traditional heuristics and rule-based methods.

However, most existing query-plan-driven systems are developed in a task-specific manner, with each system tailored to a particular task and trained from scratch. This lack of reusability significantly limits their practicality in real-world settings. Training a new model for each task is costly, especially when labeled data is difficult to obtain. For instance, collecting a training dataset for robust query optimization may take days or even months for a single query template [5], and extensive hyperparameter tuning further exacerbates inefficiency.

Despite differences in task objectives, many of these systems share a similar architecture centered around a plan embedding model [26, 43]. Typically, query plans are encoded into fixed-length vectors using tree-based models (e.g.,TreeCNN [23]), which are then passed to task-specific heads (e.g., regression for cost estimation or classification for index selection). This architectural consistency presents a natural opportunity: can we transfer knowledge across tasks by reusing the shared plan embedding model?

To explore this opportunity, a natural idea is to pretrain a query plan embedding model on a task with relatively low data collection costs, such as cost estimation, which can leverage query execution logs, and then transfer this pretrained model to downstream tasks, such as index selection or query optimization, where training effective models typically requires large volumes of labeled data.

To assess the feasibility of this idea, we consider four naive transfer strategies. These strategies begin by pretraining a plan embedding model on cost estimation, and then attempt to transfer it to a query optimization task following the setting of BAO [17]. The strategies include: (a) directly using the pretrained model without modification, (b) training a model from scratch, (c) fine-tuning

doi:10.14778/3778092.3778102

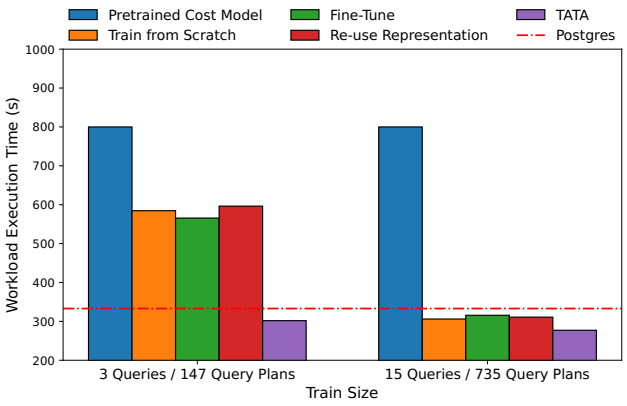


Figure 1: Performance of naive transfer strategies when adapting a cost-estimation-pretrained model to a query optimization task on the STATS dataset. Lower execution time is better.

the pretrained model using few-shot data, and (d) freezing the embedding model while training a new task-specific model.

As shown in Figure 1, none of these strategies achieves satisfactory performance. In the low-data setting (3 training queries, 147 query plans), all naive transfer strategies perform poorly. Even in the moderate-data setting (15 queries, 735 plans), none of the other three transfer strategies (a, c, d) outperforms training a model from scratch (b). These results demonstrate that simple transfer fails to offer clear benefits in terms of data efficiency or model adaptability—undermining its appeal for practical deployment in ML4DB systems.

The failure of straightforward solutions can be attributed to three fundamental challenges that hinder effective knowledge transfer:

C1. Task-Specific Optimization: Models are typically optimized for task-specific loss functions, which limits their generalizability. For example, a cost estimator focuses on predicting the latency of individual query plans, whereas query optimization requires comparing multiple plans for the same query. Thus, features learned for one task may not transfer effectively to others with different objectives.

C2. Limited Data Availability: Many ML4DB tasks require large labeled datasets that are expensive to collect. For instance, training an index selection model requires executing queries under many index configurations. In the STATS dataset, training data collection took over three months for index selection and two months for query optimization. Poor configurations also lead to long runtimes and resource overhead, exacerbating the cost.

C3. Data Distribution Shift: Transferring models across tasks inevitably introduces shifts in data distribution. For example, a cost estimator is trained on plans generated by the default optimizer, while index selection involves plans under alternative index configurations unseen during pretraining. Such distribution shifts limit transfer effectiveness.

To address these challenges, we propose TATA, a **Ta**sk **TrA**nsfer learning framework designed to enable robust and data-efficient transfer in query-plan-driven ML4DB systems. TATA is built upon two key components: *self-supervision* and *data enrichment*, which together aim to (1) leverage knowledge captured by pretrained models and (2) minimize the reliance on newly collected training data. By exploiting shared database features encoded in existing models and incorporating task-specific adaptations, TATA enables effective transfer across tasks with minimal overhead.

Self-supervision addresses Task-Specific Optimization (C1) by enhancing the generality of the plan embedding model. We propose a semi-supervised framework that integrates a self-supervised query plan reconstruction task during pretraining, enabling the model to learn transferable representations beyond cost estimation alone. To support this, we design a novel query plan decoder that reconstructs structural attributes of the input plan from its latent representation. This query plan reconstruction task acts as a regularizer, enforcing the encoder to retain fine-grained semantic and syntactic details of query plans. The resulting embeddings exhibit enhanced robustness for downstream optimization tasks, effectively bridging the gap between pretraining objectives and unseen tasks.

Data enrichment targets both Limited Data Availability (C2) and Data Distribution Shift (C3). This component synthesizes realistic, task-specific pseudo-training data by combining existing training examples with database-specific knowledge (e.g., index variations, operator choices, query rewrites). As a result, it generates sufficient training data for new tasks without requiring additional query execution, enabling scalable and efficient adaptation.

Our contributions are summarized as follows:

- This is the first work to systematically study the task transfer problem in query plan representation for learned database systems and introduce TATA, a novel framework that addresses key transferability challenges.
- We propose a new self-supervised pretraining task with a query plan decoder to improve representation robustness and generality, along with a data enrichment module that enables pseudo data generation during task transfer.
- We conduct extensive experiments to demonstrate the effectiveness of TATA. Our study covers three representative tasks (cost estimation as the source task, and query optimization and index selection as targets) and three distinct query plan representation models. We show that TATA achieves effective transfer with as few as two training queries, reducing data collection cost by up to 5× while maintaining strong performance.

2 BACKGROUND

In this section, we provide the necessary background for our study. We begin by introducing query plan representation methods, discuss their applications in ML4DB systems, and finally explore the transfer learning opportunities across tasks. Figure 2 offers a high-level illustration of this architecture.

2.1 Query Plan Representation

As illustrated in the center of Figure 2, query plan representation is a core module in a bunch of ML4DB systems spanning a variety of database tasks, such as cost estimation and query optimization. Please refer to Section 2.2 for details. This module converts a physical query plan into a vector representation which acts as the input to a machine learning model for the ML4DB task.

Existing techniques in query plan representation initially extract features from query plan nodes, such as operators, predicates, costs,

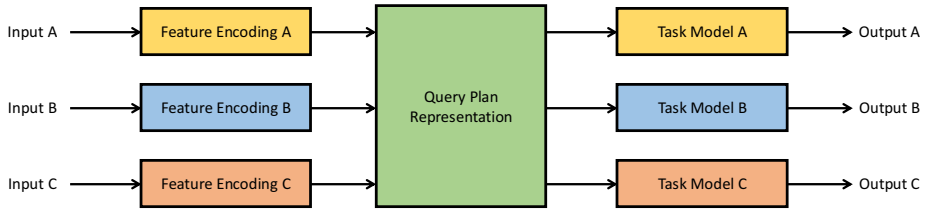


Figure 2: A unified architecture in query-plan-driven ML4DB systems. Each task (e.g., cost estimation, query optimization, index selection) extracts task-specific features from query plans and encodes them via dedicated feature encoders. The shared plan representation is then passed to task-specific models to produce outputs.

and cardinality estimates, which are derived from databases, histograms, and samples. These features are then aggregated using different representation models, including TreeCNN [17, 18, 22], TreeLSTM [30, 39], and Transformer [21, 42]. In our study, we conduct experiments with three representation models: BAO [17], QueryFormer [42] and RTOS [39], demonstrating that our proposed framework is model-agnostic and easily adaptable to any query plan representation technique.

BAO [17]. The proposed system tackles query optimization by formulating it as a multi-armed bandit problem, and proposes a reinforcement learning approach to solve the problem. In its query plan representation model, it first extracts the operator, estimated cardinality, and estimated cost from each node in a query plan, then aggregates the node features into a vector using a TreeCNN model.

QueryFormer [42]. QueryFormer is proposed as a general-purpose query plan representation model to tackle a variety of tasks. Its representation model first extracts the operator, table, predicates, and statistics features from each node. Next, it aggregates the features using a tree-structured transformer model.

RTOS [39]. RTOS is originally proposed as a reinforcement-learning-based framework for join order selection. Its representation model first learns embeddings for columns, which are augmented with predicate encodings and pooled into table representations. These table-level features are then aggregated into query plan representations using a custom TreeLSTM model that propagates information through the join tree.

2.2 Evaluating Tasks

As shown from left to right in Figure 2, each ML4DB task defines its own input format and feature encoding strategy, which are then passed through a similar query plan representation model and task-specific prediction heads. In this paper, we focus on three representative tasks: cost estimation [30], query optimization [17], and index selection [4], while acknowledging broader use cases such as join order selection [39], concurrent query optimization [21], and semantic equivalence detection [7].

Cost Estimation. This task involves predicting the resource usages of a query execution plan, typically in the form of latency as the estimation objective [30]. It involves feature extraction from each node in a query plan, feature aggregation into a single vector representation using tree-based models, and using a multi-layer perceptron (MLP) to estimate execution time.

Query Optimizer. This task involves generating a query execution plan from a SQL query. It is naturally an NP-hard problem due to the exponential search space of join orders and access methods [18]. We consider the state-of-the-art (SOTA) lines of learned optimizer research [17] with uses predefined hints to impact the master optimizer and generate a set of candidate query plans, then use a value network to choose the plan with the smallest cost.

Index Selection. This task aims to find an optimal set of indexes or index configuration for a given workload to optimize performance by reducing costs or runtime. Following the state-of-the-art strategy [4], it compares respective query plans chosen by the optimizer under each index configuration and selects the configuration that results in faster query plans. It constructs query plan pairs, encodes them into feature vectors, and predicts the faster query plan using a classifier model. The fastest index configuration is selected accordingly.

2.3 Transfer Learning Opportunities

As illustrated holistically in Figure 2, different tasks apply their own feature encoders and output heads, but share the same query plan representation model at the core. This shared architecture presents an exciting opportunity for transfer learning: ideally, a representation model pretrained on one task could generalize to others.

However, as discussed in Section 1 and shown in Figure 1, naive transfer strategies often fail to yield improvements. This highlights the need for more principled approaches, like our proposed TATA framework, explicitly addressing task-specific optimization, limited data, and data distribution shifts.

3 SYSTEM OVERVIEW

In this section, we describe our problem definition, and then give an overview of our approach.

3.1 Problem Statement

To effectively discuss our study, we introduce key terms and notations that form the foundation of our study:

(1) **Representation Model**: This model processes a physical query execution plan and converts it into an embedding vector. This embedding vector preserves important information in the feature space, enabling sharing across various tasks.
(2) **Task Model**: A machine learning model that uses query plan representation vectors to execute specific tasks, such as cost estimation or index selection.

(3) **Base Task**: The primary task from which knowledge is transferred. In our study, we focus on cost estimation as the base task.
(4) **Target Task**: The task that knowledge is transferred to, e.g., index selection or query optimization.
(5) **Source Dataset**: The labeled dataset for the base task, typically easy to collect and large.
(6) **Target Dataset**: The labeled dataset for the target task, hard to collect and significantly smaller than the source dataset.
(7) **Generated Dataset**: An artificially generated dataset for the target task, created without actual labels from execution.
(8) **Pseudo Label**: Labels generated based on domain expertise and insights from other datasets, applied to the generated dataset.

Problem Statement. Given a query plan representation model initially trained for a source task (such as cost estimation) and a considerably smaller target dataset for a new target task, the objective is to effectively adapt the representation model and train the task model. The goal is to achieve optimal task performance with minimal new training data for a target task.

3.2 Workflow

We illustrate the overall workflow of our proposed framework in Figure 3, which consists of two key stages: *Pretraining* and *Transfer to New Task*. The figure highlights our two novel components: Decoder Model (used in self-supervised learning) and the Dataset Enricher (used in data generation for transfer) in red color.

Self-supervision for Pretraining. Self-supervision has demonstrated effectiveness in improving representation quality in various domains such as natural language processing [37] and computer vision [10]. Inspired by these successes, we introduce self-supervision into query plan representation learning, enabling more robust and generalizable embeddings that directly address *C1: Task-Specific Optimization*.

As shown in the upper part of Figure 3, during the *pretraining* phase, the model is trained in a semi-supervised manner. That is, in addition to the base task objective (e.g., cost estimation), we introduce a Decoder Model that reconstructs the original query plan from its latent representation as a self-supervised learning objective. The reconstruction loss serves a dual purpose: (1) it encourages the encoder to preserve rich structural and semantic information of the query plan, and (2) it regularizes the representation learning process to enhance transferability.

Importantly, the decoder is only used during pretraining and is discarded during transfer. The component that is reused is the *Query Plan Representation model* (i.e., the encoder), which benefits from both losses. More details are provided in Section 4.

Data Enrichment for Task Transfer. In the transfer phase (shown in the lower part of Figure 3), our process begins by reusing the pretrained query plan representation model and fine-tuning it on an enhanced target dataset. As discussed in Section 1, both strategies, direct reuse and fine-tuning with limited data, tend to yield suboptimal results. This is primarily due to two key challenges: *C2: Limited Data Availability* and *C3: Data Distribution Shift*.

To address these issues, we propose a novel *Dataset Enricher* component aimed at enabling more effective and data-efficient task transfer. This module generates a large and diverse set of synthetic query plans, with the flexibility to produce arbitrary quantities in order to better cover the target query space (*C2*). Instead of relying on costly query executions for labeling, we assign *pseudo labels* to these plans using a combination of existing query execution statistics and basic operator-level domain knowledge. These operator-level semantics are generally stable across workloads, which helps the enriched dataset remain applicable in out-of-distribution (OOD) settings (*C3*).

By supplementing the limited target data with additional pseudo-labeled examples, this enrichment process helps reduce the reliance on expensive supervision and mitigates the mismatch between source and target distributions. We describe the enrichment process in more detail in Section 5.

4 SEMI-SUPERVISED PRETRAINING

In this section, we introduce the design of our self-supervised representation learning component. We begin with an overview of the motivation and architecture in Section 4.1. Section 4.2 formulates the query plan generation problem as a structured self-supervised objective. Section 4.3 presents our decoder model architecture. Finally, Section 4.4 describes how we integrate this component into the base task of cost estimation.

4.1 Overview and Motivation

We aim to improve the transferability of query plan representations by introducing a self-supervised learning objective based on plan reconstruction. To this end, we adopt an encoder–decoder architecture, where the encoder (e.g., TreeCNN [17] or QueryFormer [42]) encodes a query plan into a latent representation, and the decoder learns to reconstruct the original query plan from this embedding.

While prior learned optimizer systems [17, 18, 38, 44] are capable of generating query execution plans, they are not suitable as decoders in our setting. First, they take SQL queries rather than query plan embeddings as input. Second, their plan generation process relies on reinforcement learning, and does not provide a fully differentiable feedback, which is essential for a self-supervised training objective.

To address this gap, we formulate the query plan generation task as a sequence prediction problem that captures the hierarchical structure of the plan (Section 4.2). We then introduce a decoder that is trainable in an end-to-end manner and capable of modeling both the structural and operational components of query plans (Section 4.3). Finally, we integrate this self-supervised component into the cost estimation task to encourage the encoder to learn both structural semantics and task-relevant execution features.

4.2 Query Plan Generation as Self-supervised Objective

Problem Definition. We formulate the query plan generation task as a self-supervised learning objective, where the goal is to reconstruct a query plan from its latent vector representation. In ML4DB systems, such reconstruction not only complements representation learning, but also encourages the encoder to capture structural information that is potentially useful for downstream task transfer.

To effectively support self-supervised plan generation, the following requirements are desirable:

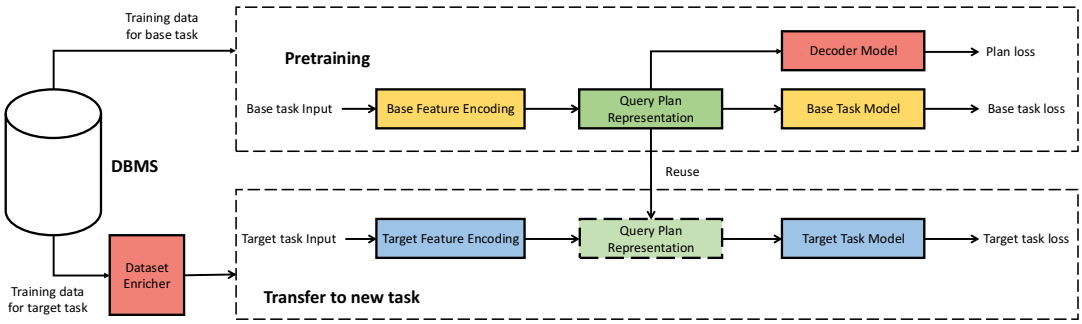
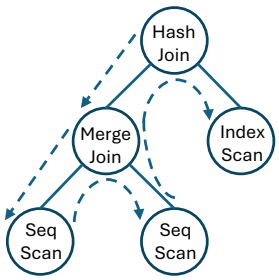


Figure 3: System workflow



Steps	Action
0	START
1	Hash Join
2	Branching
3	Merge Join
4	Branching
5	Seq Scan
6	Seq Scan
7	De-branching
8	De-branching
9	Index Scan
10	END

Figure 4: A Query Plan Generation Example. The table on the right shows the sequence of actions for generating the query plan on the left. Orange rows denote syntax actions, while blue rows represent operator tokens.

- **R1, End-to-end trainability:** The generation process must be differentiable to allow joint optimization with supervised objectives.
- **R2, Tree structure preservation:** Since query plans are inherently hierarchical, the generation model must reflect both syntactic correctness and tree structure.
- **R3, Model-agnostic compatibility:** The generation model should support integration with a wide range of representation models without architectural changes.

To address the above criteria, we propose an autoregressive query plan generation model, which is never used in query plan representation learning. Different from existing reinforcement learning-based approaches [17, 18, 38], which rely on discrete search procedures, our method formulates plan generation as a conditional sequence prediction problem. This formulation is fully differentiable, allowing backpropagation through the entire pipeline and seamless integration with neural encoders, thereby satisfying **R1 (end-to-end trainability)**.

However, query plans exhibit a hierarchical, tree-structured format, which is not naturally supported by standard autoregressive models designed for linear sequences. To address this, we define a structured action space that captures both syntax and operator-level semantics of query plans:

- **Syntax actions:** Control the structure of the plan tree using tokens such as `START`, `Branching`, `De-branching`, and `END`.
- **Operator tokens:** Specify concrete query operators (e.g., `Hash Join`, `Index Scan`) for the plan nodes.

An example of this generation process is illustrated in Figure 4. The left side shows a sample query plan, and the right table lists the corresponding generation steps. The process begins with a `START` token and performs a depth-first traversal over the plan. `Branching` tokens are emitted when entering child nodes, and `De-branching` tokens signal a return to the parent node. This structured token sequence preserves the hierarchical semantics of query plans and addresses **R2 (tree structure preservation)**.

Formally, the generation is modeled as a conditional probability distribution:

$$p(y|x) = \prod_{t=1}^{T} p(a_t \mid x, a_{<t}) \tag{1}$$

where x is the input latent representation, y is the target action sequence, and a_t denotes the action at time step t. This formulation captures both structural and sequential dependencies.

The training objective minimizes the reconstruction loss over the action sequence:

$$\mathcal{L}_{\text{recon}} = \sum_{t=1}^{T} \mathcal{L}_{\text{CE}}(a_t, \hat{a}_t) \tag{2}$$

where $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss and $\hat{a}_t$ is the predicted distribution over the action space at time step t. This self-supervised objective allows the encoder to learn transferable representations while maintaining syntactic fidelity.

Since our decoder only relies on the latent representation vector and is decoupled from specific encoder architectures, it can be directly applied to different upstream models such as TreeCNN [17] or QueryFormer [42], thus meeting **R3 (model-agnostic compatibility)**.

4.3 Decoder Model Architecture

We now describe the architecture of our query plan decoder, which is designed to be model-agnostic and compatible with a variety of representation models. The architecture is illustrated in Figure 5, and consists of three main components: a recurrent backbone, a

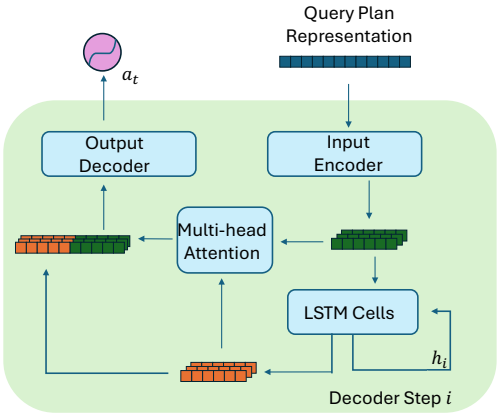


Figure 5: Decoder Model Architecture.

multi-head attention mechanism, and an output prediction layer with embedding reuse.

Given an input query plan representation vector, the decoder first projects it into a hidden vector Z that aligns with the internal state space of the decoder. This hidden vector is repeatedly fed into a recurrent neural network (RNN) to simulate the autoregressive generation process. At each time step i, the RNN produces a hidden state O_i, which is combined with Z using a multi-head attention mechanism to derive a context vector. The concatenated output is then passed to a token prediction layer to predict the next action.

Recurrent Backbone. Autoregressive generation requires the model to predict each action conditioned on the previous actions. To support this, we adopt an RNN-based architecture and specifically choose the ONLSTM [28] model, which is designed to capture tree-structured dependencies in a soft and differentiable manner.

Multi-head Attention. To enhance the decoder's ability to leverage different parts of the plan representation, we integrate a multi-head attention mechanism between the representation vector Z and the RNN outputs.

Action Token Prediction. For action prediction, we adopt an embedding reuse strategy that shares token embeddings between the encoder and decoder. Specifically, the same embedding matrix is used for both input encoding and output classification, reducing the number of parameters and improving representation consistency across tasks. This design supports efficient learning and contributes to robust and generalizable representations.

4.4 Integration with Base Task

Our self-supervised reconstruction objective can be seamlessly integrated into supervised training pipelines due to its end-to-end differentiability. We demonstrate this integration using cost estimation as the base task. During training, the decoder is attached to the final query plan representation to compute a reconstruction loss, which is then jointly optimized with the base task loss.

The overall training objective is a weighted sum of the two losses:

$$\mathcal{L}_{\text{final}} = \lambda \mathcal{L}_{\text{recon}} + (1 - \lambda)\mathcal{L}_{\text{base\_task}} \tag{3}$$

Here, $\lambda \in [0, 1]$ is a balancing coefficient that controls the contribution of the reconstruction loss relative to the base task objective. A higher value of λ encourages the model to focus more on preserving structural semantics, while a lower value emphasizes task-specific performance.

5 DATA ENRICHMENT

In this section, we introduce the design of our data enrichment module, which enhances task transfer by generating synthetic training data based on a small amount of labeled examples.

5.1 Overview

The data enrichment module is designed to address two key challenges in task transfer: *C2. Limited Data Availability* and *C3. Data Distribution Shift*. Instead of relying solely on extensive ground-truth execution data, we leverage a small set of labeled query plans to synthesize a much larger, diverse training dataset. This enriched data aims to fill the coverage gaps between source and target workloads.

The enrichment process consists of two main components:

(1) *Query Plan Generation*: We construct synthetic query plans that augment the target workload, using task-aware configurations (e.g., index setups, optimizer hints). These plans are selected to cover underrepresented query patterns and to align with task-specific requirements such as those in index selection or query optimization.
(2) *Pseudo Labeling*: To avoid expensive execution, we assign pseudo labels to the generated query plans using a two-phase method. First, we learn operator-level performance formulas based on real workload data and domain knowledge. Then, we apply these models to estimate plan-level costs via bottom-up inference.

Together, these components generate a realistic and diverse pseudo-labeled dataset that supports effective fine-tuning of the pretrained plan representation model with minimal supervision.

5.2 Workload Generation

This component aims to generate a set of synthetic query plans that (1) bridge the *distribution gap* between source and target workloads, and (2) satisfy *task-specific requirements*. To this end, we employ an importance sampling strategy guided by workload characterization, followed by task-aware plan construction.

Importance Sampling. Comprehensive data collection is often infeasible, leading to undercoverage of certain query dimensions (e.g., tables, predicates). Simple random sampling fails to ensure adequate diversity. To address this, we adopt an importance sampling strategy that prioritizes queries contributing most to minimizing the *distribution gap*—defined as the difference between feature distributions of source and target workloads.

Workload Characterization. R1D8 We represent each query workload as a feature vector summarizing key query dimensions. Specifically, we include (i) table access frequencies, (ii) column-level predicate occurrences, and (iii) join relationships between tables. These are encoded as counts over the schema graph, forming a fixed-dimensional vector for each workload. The difference between source and target vectors quantifies the *distribution gap*, which guides importance sampling.

Algorithm 1: Enriched Query Plan Generation

Input: Source dataset Q_s, target dataset Q_t, number of iterations I, number of queries per iteration N, similarity threshold θ
Output: Enriched queries Q_e, query plans P_e

1 Initialize $Q_e \leftarrow \emptyset$;
2 $F_s \leftarrow$ CharacterizeWorkload(Q_s) ;
3 **for** $i = 1$ *to* I **do**
4 $\quad F_e \leftarrow$ CharacterizeWorkload($Q_e \cup Q_t$) ;
5 $\quad F_d \leftarrow F_s - F_e$ `// Compute Distribution Differences`
6 $\quad$ **if** $F_d < \theta$ **then**
7 $\quad\quad$ **break**;
8 $\quad Q_{add} \leftarrow$ ImportanceSample(Q_s, F_d, N);
9 $\quad Q_e \leftarrow Q_e \cup Q_{add}$;
10 $P_e \leftarrow \emptyset$;
11 Apply Task requirements to *DBMS*;
12 **for** q *in* Q_e **do**
13 $\quad P_e \leftarrow P_e \cup Explain(q)$
14 **Return** Q_e, P_e;

Query Plan Generation. After selecting important queries, we construct corresponding physical plans tailored to the downstream task. For query optimization, we apply predefined hints (e.g., in BAO) to generate plan alternatives. For index selection, we simulate different index configurations via "what-if" analysis and retrieve plans using the optimizer's 'EXPLAIN'.

Algorithm. Algorithm 1 outlines the process. Starting from source and target datasets, the algorithm iteratively computes their feature differences and samples queries accordingly (lines 2–9). Once sufficient queries are collected, task-specific configurations are applied, and the optimizer generates the corresponding plans (lines 11–13).

5.3 Pseudo Labeling

Executing query plans to obtain ground-truth labels is prohibitively expensive. We address this challenge by introducing an efficient pseudo-labeling strategy that eliminates the need for full execution.

A few straightforward alternatives exist, but all exhibit notable limitations. One option is to use cost estimates directly from PostgreSQL. While these estimates are grounded in well-established heuristics, their accuracy heavily depends on cardinality estimation, which is known to be workload-sensitive and error-prone. Another approach is to use cardinality as a proxy for cost, as in [38]. However, this logical-level approximation fails to capture physical-level performance differences between operators.

To overcome these limitations, we propose a two-phase pseudo-labeling framework that combines domain knowledge with empirical insights from existing workloads. This framework is designed to (1) capture operator-level performance characteristics by integrating both domain and data-driven features, and (2) automatically generate accurate and robust pseudo labels for unseen query plans.

Concretely, the framework consists of two stages: *Formula Generation* and *Inference*. In the first stage, we define performance formulas for each database operator based on system implementation knowledge, and fit them using real execution data. In the second stage, these formulas are applied to unseen query plans to estimate their overall cost.

Formula Generation. We model the performance of each physical operator using a linear regression model, where the input features are derived from system-level semantics. These features capture key physical cost signals such as input cardinalities, child cardinalities, and relation sizes.

Instead of using fixed-cost formulas with hardcoded coefficients, we define *operator-specific feature structures* based on domain knowledge of database engine behaviors. For example, a `Sort` operator typically scales as $O(N \log N)$, while a `Nested Loop Join` scales as $O(N_1 \cdot N_2)$. These formulas determine the structure of the feature vector (e.g., $(N_1 \cdot N_2,\ N_1,\ N_2)$), while the actual coefficients are fully learned from real query execution data using linear regression.

Table 1: Operator-specific feature bases $\phi(\text{op})$ used for pseudo labeling. Here, N denotes the estimated cardinality of the operator, N_1 and N_2 denote the estimated output cardinalities of its child operators, and *TableSize* is the size of the referenced relation. All coefficients for these features are learned via linear regression.

R3W1, D1, Meta2

Operator Type	Feature Basis $\phi(\text{op})$
`Nested Loop Join`	$(N_1N_2,\ N_1,\ N_2)$
`Merge Join`	$(N_1 + N_2,\ N_1 \log N_1,\ N_2 \log N_2)$
`Hash Join`	$(N_1,\ N_2)$
`Sort`	$(N \log N,\ N)$
`Sequential Scan`	(*TableSize*)
`Index Scan / Bitmap Scan`	(N)
`Gather`	(N)
`Default (Fallback)`	(N_1, N_2, N_1N_2)

Algorithm 2: Fitting Operator Models

Input: Source dataset Q_s, Target dataset Q_t, formula
Output: Operator models $\mathcal{M}$

1 Initialize $\mathcal{M} \leftarrow \emptyset$;
2 $Q \leftarrow Q_s \cup Q_t$;
3 **for** *each op in operators* **do**
4 $\quad$ **if** *op in domain_knowledge* **then**
5 $\quad\quad features \leftarrow formula[op]$;
6 $\quad$ **else**
7 $\quad\quad features \leftarrow default\_features$;
8 $\quad data \leftarrow$ extractNodes(Q_s, op);
9 $\quad model \leftarrow$ fitLinear($data, features$);
10 $\quad \mathcal{M}[op] \leftarrow model$;
11 **Return** $\mathcal{M}$;

For operators not explicitly covered by domain formulas (e.g., less common or database-specific nodes), we use a fallback feature extractor that relies on basic cardinality patterns such as the node's own estimated cardinality and those of its children. This allows the model to generalize even in the presence of unseen operator types. Table 1 enumerates the feature bases $\phi(\text{op})$ used for each operator type in our pseudo-labeling framework. Note that it is flexible enough to accommodate new formula.

Inference. Once operator models are trained, we estimate the cost of a query plan via bottom-up traversal. For each operator node, we extract input features (e.g., estimated cardinality of the node and its children nodes N, N_1, N_2) obtained from the DBMS query plan (e.g., via `EXPLAIN`), without executing the query. The learned model predicts the local cost, which is combined recursively with child costs to yield the full plan cost. This allows us to generate cost estimates efficiently while correcting for biases present in the underlying cardinality estimators.

Limitations. Our pseudo-labeling method relies on operator-level cost models derived from domain knowledge and fitted using workload data. This design aligns with database systems that employ cost-based optimization and expose interpretable physical query plans (e.g., PostgreSQL, MySQL). In contrast, systems with adaptive execution (e.g., SQL Server's adaptive joins) or proprietary, black-box optimizers (e.g., Snowflake, BigQuery) often

Algorithm 3: Inference

Input: Query plan q, operator models $\mathcal{M}$
Output: Estimated cost c

$c \leftarrow$ **computeCost**(q_{root}, $\mathcal{M}$);
Return c;

Function computeCost(*node*, $\mathcal{M}$):
  $model \leftarrow \mathcal{M}[node.\text{type}]$;
  $cost \leftarrow model.\text{predict}(node.\text{features})$;
  if $node.children \neq \emptyset$ **then**
    for *each child in node.children* **do**
      $cost \leftarrow cost +$ computeCost (*child*, $\mathcal{M}$);
  Return *cost*;

lack sufficient transparency for such modeling. We leave the development of pseudo-labeling strategies for these systems to future work.

We also note that these pseudo labels are not perfect estimators. For example, operator-level regression errors may compound in complex plans, and runtime data itself can be noisy due to hardware variance. Nevertheless, their main purpose is not to act as precise cost predictors, but to serve as corrective signals to the transfer based on concrete database domain knowledge and empirical data. In this role, pseudo labels provide auxiliary supervision for transfer learning, whereas building a highly accurate learned cost estimator would require much larger training data and thus defeat the original purpose.

5.4 System Integration

The enriched dataset produced by our workload generation and pseudo-labeling modules serves as supplemental training data for target tasks. Once generated, these synthetic query plans with pseudo labels are combined with a small set of real labeled samples to fine-tune the pretrained query plan representation model. This integration enables robust task transfer with minimal reliance on costly query executions and allows the model to better adapt to new task objectives and data distributions.

6 EXPERIMENTS

To empirically evaluate the effectiveness of TATA in task transfer, we experiment the framework to two sets of task transfers. We perform pretraining of the query plan representation model by jointly train on cost estimation and query plan reconstruction. Next, we transfer from the pretrained embedding to index selection and query optimizer.

6.1 Experimental Setup

Datasets. We use four real-life datasets in our experiments for all tasks.

- **Join Order Benchmark (JOB)**. This dataset comprises of 113 queries from 33 parameterized query templates using the Internet Movie Database (IMDB) [15]. We generated 200 queries from each template with uniform sampling over predicate columns from the join results, and use the original JOB queries for testing.
- **STATS**. This real-world dataset is an anonymized dump of user-contributed content on the Stats Stack Exchange network [6]. It follows a more complex schema topologywith high skewness within each column and high correlation between tables and columns, making it particularly challenging. Following the original work, we use 70,142 generated queries for pre-training,and 146 hand-crafted queries as the test set.
- **CEB**. CEB builds upon IMDB dataset with 3000 queries from 16 templates designed to be specifically challenging [24]. We randomly split the pretraining and test dataset with nine-to-one ratio.
- **STACK**. Stack is built upon over 18 million posts from StackExchange spanning ten years [17]. It contains 5000 queries. We randomly split the pretraining and test dataset with nine-to-one ratio.

Pretraining. For each dataset, we pretrain the query plan representation model using the loss signal from both cost estimation and self-supervision. The loss ratio is controlled by hyper-parameter λ.

Baselines. We compare TATA with four classes of baselines as follows. First, we compare with using Postgres default cost estimator for all tasks. This baseline demonstrates the performance of solutions without machine learning component. Next, we compare with fine-tuning, train-from-scratch, and the original model as the three common alternative strategies for transfer learning. Finally, we include results with full-data training as a reference for the potential performance ceiling.

Setup. All experiments are conducted on a machine with an Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz with 128GB RAM, and an NVIDIA GeForce RTX 3080 GPU, with PostgreSQL 13.

6.2 Effectiveness on Learned Query Optimizer

We evaluate the effectiveness of TATA's transfer learning capabilities by applying the learned query plan representation model to a learned query optimizer using the BAO [17] framework, a widely adopted learned optimizer that has also inspired subsequent systems such as AutoSteer [2] and FASTgres [33]. Specifically, BAO estimates the performance of alternative query plans for a given query by exploring combinations of hint sets (referred to as bandit arms in a multi-armed bandit problem) and selects the best query plan to execute.

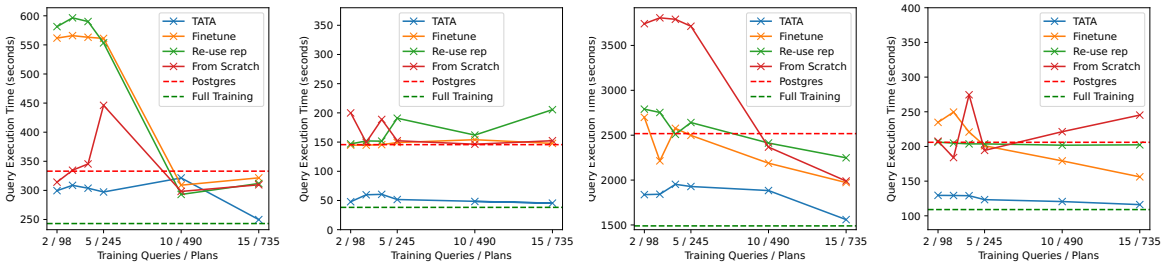
We apply the 49 default hint combinations specified in the original BAO paper [17]. We first pretrain the representation model on the cost estimation task, which does not involve varying hint settings. Next, we fine-tune the pretrained model using a limited number of examples, where each query is executed under different hint sets. Finally, we assess query optimization by measuring the cumulative execution latency for the test workload. We note that TATA only affects pretraining and transfer; that is, the inference path remains unchanged, with candidate plans enumerated and scored in a batched forward pass.

We present the few-shot transfer learning results in Figure 6. The top row represents cumulative query execution time of a query workload with respect to the number of training queries and query plans, and the bottom row depicts the dataset collection cost to obtain the labels for the training query plans. The success of a model is measured by performing well with as few training data as possible. For all datasets, we observe that TATA quickly outperforms Postgres baseline with only as few as two training queries (98 query plans), and performs consistently well as we increase training queries. It performs particularly well on IMDB and STACK datasets which shorten the query execution time by about 60%, while most other baselines struggle to perform comparable with Postgres with as many as 15 queries (735 query plans) as training data.

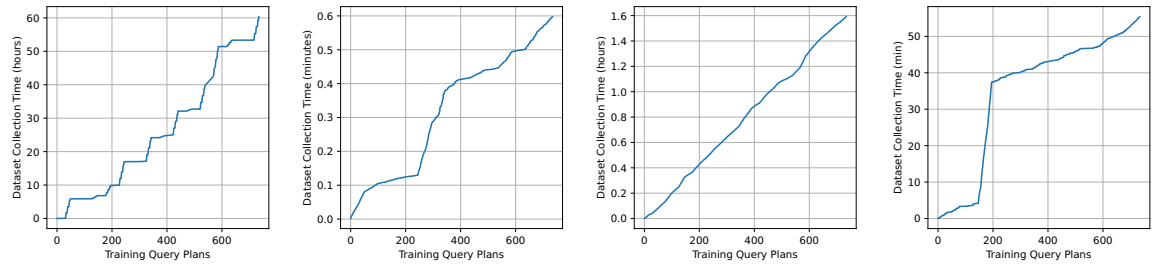
Translating to dataset collection cost, TATA saves significant dataset collection time to achieve stable and good performance. For example, it requires more than 5X shorter dataset collection time (up to 30 hours in STATS dataset) than all other methods. This demonstrates that TATA can improve the practicality of task adaptation: one can quickly experiment with new other tasks at much lower cost.

6.3 Effectiveness on Index Selection

We evaluate the transfer learning effectiveness of TATA on index selection task. We follow AIMEETSAI [4] framework, which compares index configurations by comparing the corresponding query plans of a query put in the index configuration, i.e., faster query plan execution corresponds to more optimal index configuration. In this setting, index selection is transformed into a classification problem. We use the toolkit [13] and HypoPG [27] to

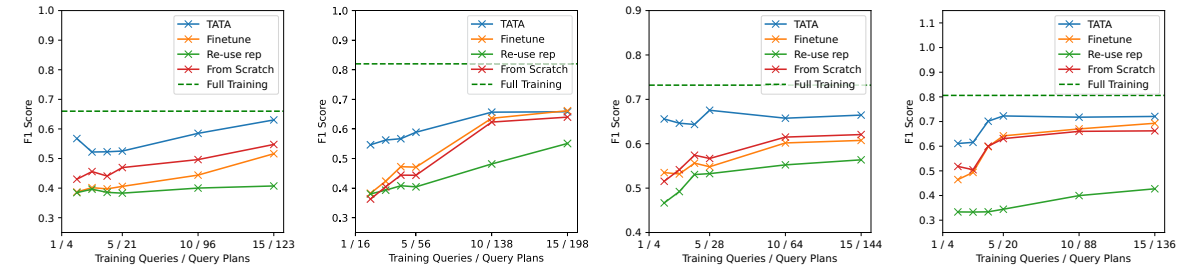


(a) STATS Test Performance. (b) IMDB Test Performance. (c) CEB Test Performance. (d) STACK Test Performance.

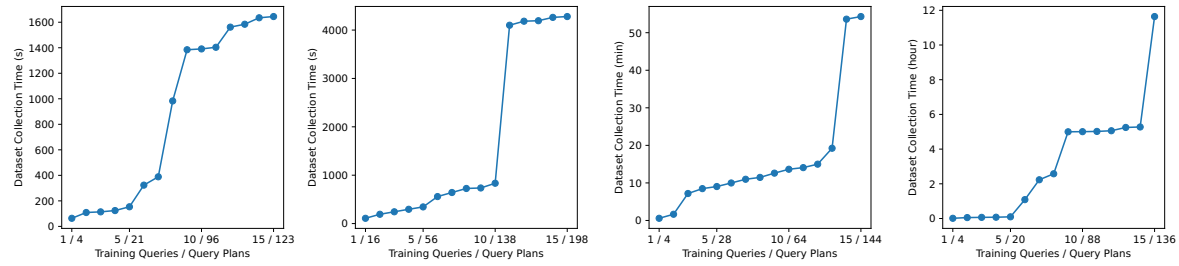


(e) STATS Data Collection Cost. (f) IMDB Data Collection Cost. (g) CEB Data Collection Cost. (h) STACK Data Collection Cost.

Figure 6: Transfer learning to learned optimizer performance and dataset collection time. The X-axis labels denote the number of training queries and the corresponding number of query plans. Each query is executed under 49 hint configurations.



(a) STATS Test Performance. (b) IMDB Test Performance. (c) CEB Test Performance. (d) STACK Test Performance.



(e) STATS Data Collection Cost. (f) IMDB Data Collection Cost. (g) CEB Data Collection Cost. (h) STACK Data Collection Cost.

Figure 7: Transfer learning to index selection performance and dataset collection time. The X-axis labels denote the number of training queries and the corresponding number of query plans. The plan counts are obtained by enumerating alternative index configurations for each query.

manage hypothetical and actual indexes. We use F1 score to measure the classification performance.

We present the results in Figure 7. We observe that TATA consistently outperforms the baselines. Specifically, in STATS, CEB and STACK datasets,

TATA performs much better than others consistently across different number of training queries, whereas in IMDB, the total number of queries required to perform well converges for some baseline methods, but TATA performs much better than them with smaller number of training data.

6.4 Analyzing Self-supervised learning

Convergence analysis. We study the training efficiency of the self-supervised learning model, and the impact of the simultaneous training with the cost estimation model. Figure 8 demonstrates the validation loss with respect to the number of epochs. As shown in Figure 8(a) and Figure 8(b), when training separately, the cost estimation model converges at about 50 epochs, whereas self-supervision model converges at about 180 epochs. However, when we train both tasks simultaneously in Figure 8(c), we observe the loss of cost estimation converges at roughly the same number of epochs, while the self-supervision model converges faster than when trained alone, at about 125 epochs. This observation suggests two findings. First, a query plan representation has sufficient representation power to fulfill the needs for both tasks. Second, the two tasks are compatible for joint training, that they can help each other for faster convergence.

Impact on cost estimation. A natural concern with introducing self-supervised learning as an additional loss term during training is the potential compromises of performance on cost estimation. To illustrate the impact, we present the cost estimation accuracy on the STATS datasets. The results on other datasets follow the same trend.

As discussed in Section 4, parameter λ controls the ratio between self-supervised learning and cost estimation: a higher λ value places more emphasis on self-supervision, while a lower value prioritizes cost estimation. Figure 9 presents the training and test errors associated with different λ values.

The results show that for λ values between 0.05 and 0.95, both the training and test Q-Errors are comparable to those obtained with λ set to 0, where the model is trained exclusively on cost estimation. This indicates that integrating self-supervised learning does not negatively impact the cost estimation task.In contrast, setting λ to 1 results in significantly higher Q-Errors. This is an expected behavior, because the cost estimation loss is entirely excluded from the training process.

Impact on transfer learning. Given the black-box nature of representation learning, we study the impact of self-supervision through an ablation study to hyper-parameter λ, which controls the proportion of reconstruction and cost estimation loss. We present the results on STATS dataset in Figure 10, where the trends on other datasets are similar. We find that intermediate values ($0.2 \leq \lambda \leq 0.8$) yield the best transfer performance, while extreme values ($\lambda \approx 0$ or $\lambda \approx 1$) degrade it. This indicates that the two objectives capture complementary aspects of query plans: reconstruction preserves structural semantics such as operator types and join patterns, whereas cost estimation emphasizes execution-related signals like cardinalities and costs. Their combination regularizes the encoder to retain both, producing more robust and transferable representations. In practice, this finding means that λ can be set to any in-between value (we use $\lambda = 0.5$ by default), and no dataset-specific tuning is required.

6.5 Analysis on Data Enrichment.

Pseudo labeling accuracy. In the data enrichment module, we label the latencies of synthetic query plans with our proposed pseudo labeling technique. We execute the synthetic query plans to collect the actual latencies, and compute the accuracy of the pseudo labels.

Following common practice in cost estimation [30, 42], we first report Q-Error, which measures the multiplicative error between the estimated value and actual value. A perfect estimation yields 1, while inaccuracies produce larger positive numbers. We additionally report absolute error in seconds. We show detailed results on the STATS dataset in Table 2, and observed similar trends on other datasets.

As seen from Table 2, pseudo labels achieve accuracy that lies between Postgres cost estimates and learned models across both metrics. This is expected, since pseudo labels are based on lightweight operator formulas rather than high-capacity learned predictors. While they are not intended to serve as precise cost estimators, they provide consistent operator-level performance patterns that act as useful auxiliary signals, helping the representation adapt more effectively during transfer.

Another important observation is that formula-based models, such as Postgres estimates and pseudo-labeling approach, are less prone to overfitting. This is reflected in their comparable accuracy on both training and test workloads. Their robustness stems from reliance on fundamental database mechanisms, such as the consistent impact of relation size on scan latency, which holds regardless of query distribution. Given their foundation in formula-based models, we consider pseudo labels a regularizer rather than an alternative cost model.

Table 2: Cost estimation accuracy.

STATS	Training Q-Error			Test Q-Error		
	Median	90%	99%	Median	90%	99%
Pseudo	6.94	130.9	487.3	10.1	164.8	566.4
Postgres	124.1	343.0	621.4	33.5	208.9	422.3
Model	1.19	3.13	31.42	2.00	28.11	196.7
STATS	Training Abs. Error (s)			Test Abs. Error (s)		
	Median	90%	99%	Median	90%	99%
Pseudo	1.15	30.5	58.2	7.99	103	1730
Postgres	13.1	231	4670	24.8	18.5	6580
Model	0.021	1.86	369	0.382	2.71	1024

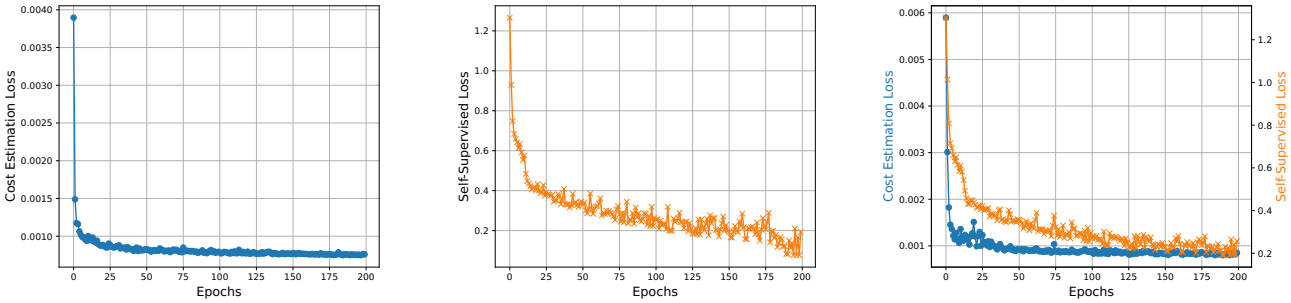
Effect of using pseudo labels directly in downstream tasks. To further clarify the role of pseudo labels, we evaluate what happens if they are used *directly* as labels for target tasks, without TATA's transfer procedure. Specifically, we make inference on the learned optimizer (BAO) and index selection model using only pseudo-labeler as a model. Table 3 reports their performance on STATS dataset, where the results on other datasets are similar.

Table 3: Using pseudo labels directly in downstream tasks on STATS datase. Lower runtime is better; higher F1 is better.

Method	BAO Runtime (s)	Index Selection F1
Pseudo labels only	549	0.45
TATA (ours)	254	0.62

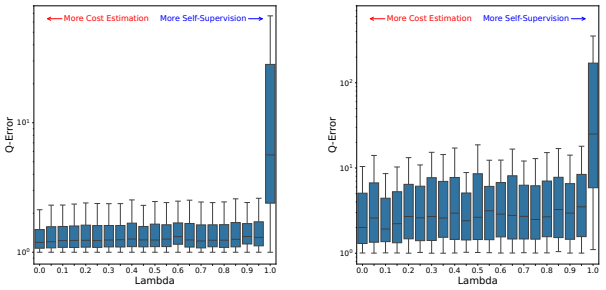
The results show that pseudo labels alone perform substantially worse than TATA. This highlights an important distinction: pseudo labels are lightweight and free to obtain, but too simplistic to replace learned cost models. Query-plan–driven learned systems are necessarily more complex and slower to train, yet they capture correlations that pseudo labels miss. TATA leverages both: it retains the expressiveness of plan-driven models while exploiting pseudo labels as auxiliary corrective signals to reduce the amount of expensive training data needed.

Varying the enriched dataset size. We study the impact of the enrichment size on task transfer effectiveness. We present the result with respect to enrich size in Figure 11 for both STATS and IMDB datasets on query optimization. We observe that for STATS datasets, we need at least about 500 enriched query plans to outperform Postgres baseline, and the score converges when we have about 1600 query plans. For IMDB, we see remarkable time reduction at only 100 query plans, and the execution time converges at about 400 query plans. Although enriched datasets with pseudo labels can be collected at minimal cost, our findings indicate that a dataset of around 1600 enriched query plans is sufficient to facilitate effective task transfer.



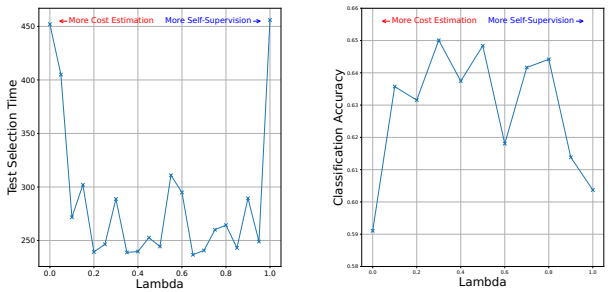
(a) Cost Estimation. (b) Self-supervision. (c) Both.

Figure 8: Convergence Analysis for Pretraining with self-supervision.



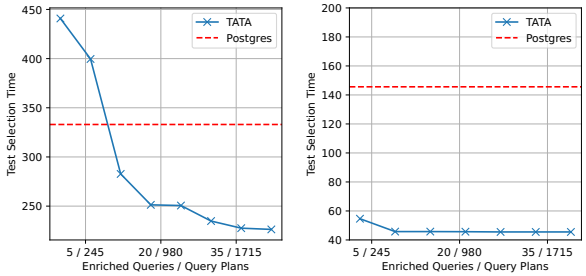
(a) Training Accuracy. (b) Test Accuracy.

Figure 9: Impact of self-supervision on cost estimation pretraining performance.



(a) Query Optimization. (b) Index selection.

Figure 10: Impact of self-supervision on transfer learning performance.



(a) STATS. (b) IMDB.

Figure 11: Impact of enrichment size on transfer learning performance.

Table 4: Efficiency of pipeline. Times in seconds (wall clock).

Stage	Breakdown Item	CPU (s)	GPU (s)
Pretrain	w/o self-supervision	168	111
	with self-supervision	1560	1200
Data Enrich	Formula Fitting	2.17	—
	Sampling (20 queries)	0.30	—
	Plan Generation (980 plans)	2.58	—
	Pseudo Labeling (980 plans)	0.82	—
	total	**5.87**	—
Transfer Training	w/o enrichment	0.49	0.33
	Enrichment + training	7.25	6.92

6.6 Efficiency Analysis

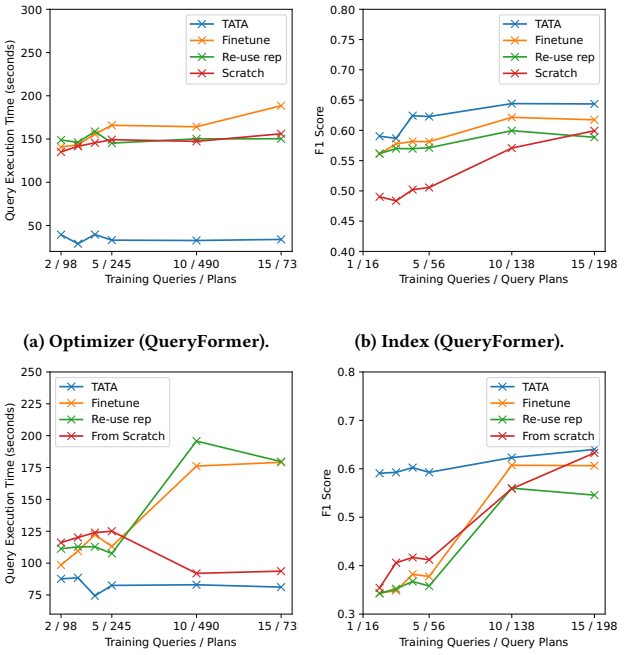
We analyze the computational efficiency of TATA on both pretraining and transfer learning stages using the STATS dataset. Table 4 reports wall-clock times on CPU and GPU.

First, we observe that self-supervision slows pretraining by about 10 times, but this cost is paid once and the model can be reused across tasks. Second, for data enrichment, generating ~1,000 pseudo-labeled plans takes under 6 seconds on CPU and ~10 MB of storage. The main cost lies in invoking the `EXPLAIN` command whereas pseudo-labeling itself is lightweight. Enrichment is thus efficient and practically bounded. Third, for transfer learning, we report *enrichment + training* as the total transfer cost. With enrichment the pipeline remains under 8 seconds on CPU (still practically usable), while without enrichment transfer completes in under 1 second. Last, we note that TATA modifies pretraining and transfer only; the inference path is unchanged relative to the chosen query plan encoder (e.g., BAO/QueryFormer), while the The decoder is *not* used at inference.

6.7 Is TATA model-agnostic?

The design choices of TATA framework, i.e., self-supervision and data enrichment, are both model-agnostic with respect to the choice of underlying query plan representation models. To test for the model-agnostic capacity, we experiment with another two query plan representation models, QueryFormer [42] and RTOS [39], which are popular query plan representation models based on transformer and Tree LSTM models respectively [43]. We perform similar experiments as with BAO's setting, on both query optimization and index selection tasks, and present the results on IMDB dataset in Figure 12. We note that the results on other datasets follow a similar trend.

Similar to the observations as with BAO's representation model, we observe that TATA can outperform all the baselines especially with less traning data. For query optimization, TATA achieves 3x speedup with merely two queries and less than 100 query plans as training data. For index selection, TATA achieves an average improvement of about 4% in F1 score over the second-best baseline across all training sizes.



(a) Optimizer (QueryFormer). (b) Index (QueryFormer).

(c) Optimizer (RTOS). (d) Index (RTOS).

Figure 12: Task Transfer Performance using other representation models.

7 RELATED WORK

Query plan representation learning in learned data systems. Learned workload-driven database systems have achieved great success in the past decade by leveraging query execution logs of workloads to solve difficult database problems, including cardinality estimation [12], join order selection [40], cost estimation [11, 30], index selection [4, 29], view recommendation [41], query optimizer [17, 18, 38], and many others [8]. Among these, a large number of proposals use query plans as input. A common first step among these proposals is to extract important features from query plan nodes, and learn a vector representation for the query plans for downstream models [43]. Methods to adapt different tree models are innovated to handle the tree structures of the input, including TreeCNN [17], TreeLSTM [30], LSTM [41], TreeRNN [1], Feature Vectors [4], Transformers [21, 42]. Some study explore the multi-task capacity [30, 42, 43] and pretrained models [26].

In addition to task-specific systems, some works explicitly focus on representation learning itself. Marcus and Papaemmanouil [19] propose a method to learn operator-level embeddings based on contextual execution statistics. Ba and Rigger [3] propose a unified intermediate representation to support cross-system compatibility by identifying shared abstractions in query plan trees across multiple database engines. While their representation facilitates tool portability and system-agnostic analysis, it is not designed for task-level transfer of learned models to support cross-system reasoning and transferability. Tsapelas and Koutrika [32] develop QPSeeker, a variational inference-based neural planner that simultaneously models query content and data properties to produce holistic plan representations. An autoencoder-based framework [31] proposes to use self-supervised plan embeddings for query plan search, illustrating the utility of learned latent spaces for tasks beyond supervised learning. Unlike our method, their self-supervision is used solely for offline plan ranking rather than enabling transfer across tasks. While several of these methods contribute meaningful progress in representation quality and modeling flexibility, transfer learning across tasks remains rarely studied in a principled way. This work addresses this important gap, proposing a task transfer framework that enables robust reuse of query plan encoders across diverse ML4DB problems.

Transfer learning in database systems. In recent years, multiple efforts have examined the transfer problem in learned database systems. A line of research studies the workload shift, i.e., adapting models to unseen query regions. DDUp [14] proposes to automatically detect workload drift and then applies transfer techniques such as fine-tuning, distillation, or retraining. Warper takes a generative route, using GANs to synthesize additional queries for more efficient adaptation of learned cardinality estimators to both data and workload drifts [16]. Robust MSCN addresses out-of-distribution cardinality estimation by introducing query masking, which hides features during training to encourage reliance on DBMS estimates [25]. ShiftHandler [34] instead employs a replay buffer to capture workload shifts and OOD cases. Finally, Wu et al. [35] provide a theoretical perspective on the generalization capacity of query-driven selectivity models, introducing a framework that relaxes standard probability-based assumptions and broadens applicability.

Another line of research proposes to use the concept of foundation models for broader generalization across arbitrary datasets and database tasks. Hilprecht and Binnig [9] introduce a novel concept of zero-shot learning for database systems. They propose a workload-agnostic encoding scheme so that a model can adapt to unseen database schema out-of-the-box without retraining on new databases. Wu et al. [36] propose a unified transferable model (MTMLF) that employs a multi-task training approach to capture transferable knowledge across tasks and a pre-train fine-tune procedure to distill transferable knowledge across databases.

Existing studies often focus on enhancing cardinality estimation or propose broad frameworks aimed at generalization. However, these are not tailored specifically and directly applicable to the task of transferring query plan representations. This work proposes a nuanced approach that not only addresses specific transfer challenges in learned database systems but also aims to set the foundation for evolving towards more robust, universally applicable models in future developments.

8 LIMITATIONS AND FUTURE WORK

While our study demonstrates the effectiveness of TATA across three representative tasks (cost estimation, query optimization, and index selection) and three plan representation models, it does not encompass the full range of ML4DB applications. Additional tasks such as view recommendation and semantic equivalence detection remain to be explored. Moreover, our evaluation is conducted on static benchmarks to ensure reproducibility and fair comparisons, whereas real-world databases evolve over time with changing workloads, schemas, and concurrency demands. Extending TATA to operate in such dynamic settings is an important direction for future work. We view our work as a step toward more general and reusable query plan representations, providing a foundation for subsequent extensions to broader tasks and deployment environments.

9 CONCLUSION

This paper presents TATA, a novel framework for the task transfer problem in query-plan-driven ML4DB systems. It effectively reduces the amount of the data size required for new data collection by employing self-supervised learning and pseudo labeling techniques. Through extensive experiments, we demonstrate TATA's ability to deliver substantial improvements in both performance and transfer efficiency, across three query plan representation models and three representative tasks.

ACKNOWLEDGMENTS

This research is supported in part by Alibaba Talent Program and Singapore MOE AcRF Tier-2 grant MOE-T2EP20223-0004.

REFERENCES

[1] Mert Akdere, Ugur Çetintemel, Matteo Riondato, Eli Upfal, and Stanley B. Zdonik. 2012. Learning-based Query Performance Modeling and Prediction. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*. IEEE Computer Society, 390–401.

[2] Christoph Anneser, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, and Ryan Marcus. 2023. AutoSteer: Learned Query Optimization for Any SQL Database. *Proc. VLDB Endow.* 16, 12 (2023), 3515–3527.

[3] Jinsheng Ba and Manuel Rigger. 2025. Towards a Unified Query Plan Representation. arXiv:2408.07857 [cs.SE]

[4] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2019. AI Meets AI: Leveraging Query Executions to Improve Index Recommendations. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1241–1258.

[5] Lyric Doshi, Vincent Zhuang, Gaurav Jain, Ryan Marcus, Haoyu Huang, Deniz Altinbüken, Eugene Brevdo, and Campbell Fraser. 2023. Kepler: Robust Learning for Parametric Query Optimization. *Proc. ACM Manag. Data* 1, 1 (2023), 109:1–109:25.

[6] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2022. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proc. VLDB Endow.* 15, 4 (2022), 752–765.

[7] Brandon Haynes, Rana Alotaibi, Anna Pavlenko, Jyoti Leeka, Alekh Jindal, and Yuanyuan Tian. 2023. GEqO: ML-Accelerated Semantic Equivalence Detection. *Proc. ACM Manag. Data* 1, 4 (2023), 223:1–223:25. https://doi.org/10.1145/3626710

[8] Brandon Haynes, Rana Alotaibi, Anna Pavlenko, Jyoti Leeka, Alekh Jindal, and Yuanyuan Tian. 2024. GEqO: ML-Accelerated Semantic Equivalence Detection. *CoRR* abs/2401.01280 (2024).

[9] Benjamin Hilprecht and Carsten Binnig. 2021. One Model to Rule them All: Towards Zero-Shot Learning for Databases. *ArXiv* abs/2105.00642 (2021).

[10] Longlong Jing and Yingli Tian. 2021. Self-Supervised Visual Feature Learning With Deep Neural Networks: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 43, 11 (2021), 4037–4058.

[11] Johan Kok Zhi Kang, Gaurav, Sien Yi Tan, Feng Cheng, Shixuan Sun, and Bingsheng He. 2021. Efficient Deep Learning Pipelines for Accurate Cost Estimations Over Large Scale Query Workload. In *SIGMOD*. ACM, 1014–1022.

[12] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2018. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. *CoRR* abs/1809.00677 (2018).

[13] Jan Kossmann, Stefan Halfpap, Marcel Jankrift, and Rainer Schlosser. 2020. Magic Mirror in My Hand, Which is the Best in the Land? An Experimental Evaluation of Index Selection Algorithms. 13, 12 (2020).

[14] Meghdad Kurmanji and Peter Triantafillou. 2023. Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data. *Proc. ACM Manag. Data* 1, 1 (2023), 27.

[15] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215.

[16] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1920–1933.

[17] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. *Bao: Making Learned Query Optimization Practical*. Association for Computing Machinery, New York, NY, USA, 1275–1288.

[18] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (2019), 1705–1718.

[19] Ryan Marcus and Olga Papaemmanouil. 2019. Flexible Operator Embeddings via Deep Learning. *CoRR* abs/1901.09090 (2019).

[20] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *Proc. VLDB Endow.* 12, 11 (July 2019), 1733–1746.

[21] Songsong Mo, Yile Chen, Hao Wang, Gao Cong, and Zhifeng Bao. 2023. Lemo: A Cache-Enhanced Learned Optimizer for Concurrent Queries. *Proc. ACM Manag. Data* 1, 4 (2023), 247:1–247:26.

[22] Songsong Mo, Yue Zhao, Zhifeng Bao, Quanqing Xu, Chuanhui Yang, and Gao Cong. 2024. RankPQO: Learning-to-Rank for Parametric Query Optimization. *Proceedings of the VLDB Endowment* 18, 3 (2024), 863–875.

[23] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press.

[24] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-loss: learning cardinality estimates that matter. *Proc. VLDB Endow.* 14, 11 (July 2021), 2019–2032.

[25] Parimarjan Negi, Ziniu Wu, Andreas Kipf, Nesime Tatbul, Ryan Marcus, Sam Madden, Tim Kraska, and Mohammad Alizadeh. 2023. Robust Query Driven Cardinality Estimation under Changing Workloads. *Proc. VLDB Endow.* 16, 6 (2023), 1520–1533.

[26] Debjyoti Paul, Jie Cao, Feifei Li, and Vivek Srikumar. 2021. Database Workload Characterization with Query Plan Encoders. *Proc. VLDB Endow.* 15, 4 (2021), 923–935.

[27] Julien Rouhaud and contributors. 2024. HypoPG: Hypothetical indexes for PostgreSQL. https://github.com/HypoPG/hypopg

[28] Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron C. Courville. 2018. Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks. *CoRR* abs/1810.09536 (2018).

[29] Jiachen Shi, Gao Cong, and Xiaoli Li. 2022. Learned Index Benefits: Machine Learning Based Index Performance Estimation. *Proc. VLDB Endow.* 15, 13 (2022), 3950–3962.

[30] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *PVLDB* 13, 3 (2019), 307–319.

[31] Jeffrey Tao, Natalie Maus, Haydn Thomas Jones, Yimeng Zeng, Jacob R. Gardner, and Ryan Marcus. 2025. Learned Offline Query Planning via Bayesian Optimization. *Proc. ACM Manag. Data* 3, 3 (2025), 179:1–179:29.

[32] Christos Tsapelas and Georgia Koutrika. 2024. QPSeeker: An Efficient Neural Planner combining both data and queries through Variational Inference. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*, Letizia Tanca, Qiong Luo, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani (Eds.). OpenProceedings.org, 307–319.

[33] Lucas Woltmann, Jerome Thiessat, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. 2023. FASTgres: Making Learned Query Optimizer Hinting Effective. *Proc. VLDB Endow.* 16, 11 (2023), 3310–3322.

[34] Peizhi Wu and Zachary G Ives. 2024. Modeling Shifting Workloads for Learned Database Systems. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–27.

[35] Peizhi Wu, Haoshu Xu, Ryan Marcus, and Zachary G. Ives. 2024. A Practical Theory of Generalization in Selectivity Learning. arXiv:2409.07014

[36] Ziniu Wu, Peilun Yang, Pei Yu, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2021. A Unified Transferable Model for ML-Enhanced DBMS. *CoRR* abs/2105.02418 (2021).

[37] Yutao Xie, Qiyu Wu, Wei Chen, and Tengjiao Wang. 2022. Stable Contrastive Learning for Self-Supervised Sentence Embeddings With Pseudo-Siamese Mutual Learning. *IEEE ACM Trans. Audio Speech Lang. Process.* 30 (2022), 3046–3059.

[38] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a Query Optimizer Without Expert Demonstrations. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 931–944. https://doi.org/10.1145/3514221.3517885

[39] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement Learning with Tree-LSTM for Join Order Selection. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 1297–1308.

[40] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement Learning with Tree-LSTM for Join Order Selection. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 1297–1308.

[41] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. 2020. Automatic View Generation with Deep Learning and Reinforcement Learning. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 1501–1512.

[42] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. QueryFormer: A Tree Transformer Model for Query Plan Representation. *Proc. VLDB Endow.* 15, 8 (2022), 1658–1670.

[43] Yue Zhao, Zhaodonghui Li, and Gao Cong. 2024. A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. *Proc. VLDB Endow.* 17, 4 (2024), 823–835.

[44] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A Learning-to-Rank Query Optimizer. *Proc. VLDB Endow.* 16, 6 (2023), 1466–1479.