DACE: A Database-Agnostic Cost Estimator

Zibo Liang[1], Xu Chen[1], Yuyang Xia[1], Runfan Ye[1], Haitian Chen[1], Jiandong Xie[2], Kai Zheng[1,*]
[1]University of Electronic Science and Technology of China, China [2]Huawei Technologies Co., Ltd., China
{zbliang, xuchen, xiayuyang, yerunfan, haitianchen}@std.uestc.edu.cn,
xiejiandong@huawei.com, zhengkai@uestc.edu.cn

***Abstract*—Cost estimation is of great importance in query optimization. However, traditional optimizers compute the cost based on heuristics, sacrificing accuracy for efficiency. In recent years, learning-based cost estimation models have achieved high accuracy. However, their poor robustness and inefficiency lead to their failure to meet the needs of practical scenarios. We propose a lightweight and Database-Agnostic Cost Estimation model (*DACE*) to address the above limitations. To further improve the effectiveness of *DACE*, we design a tree-structure-based loss adjustment strategy to learn sub-plan information and solve the information redundancy problem. As a pre-trained estimator, *DACE* can efficiently make accurate predictions on unseen databases. For more complex scenarios, we fine-tune *DACE* with LoRA. The excellent efficiency allows *DACE* to adapt to challenging scenarios with minimal effort. As a pre-trained encoder, *DACE* can improve the accuracy and robustness of other cost estimation models through knowledge integration and solve the notorious cold start problem. Extensive experiments have shown that *DACE*'s accuracy, efficiency, and robustness are much better than existing methods.**

***Index Terms*—Cost estimation, Query optimization**

I. Introduction

Cost estimation has a wide range of applications in many areas, such as resource allocation [28] and query optimization [11], [17]. The traditional database optimizer compute the cost of query plans based on heuristics [36], ensuring efficiency but sacrificing accuracy. In recent years, many researchers have attempted to apply machine learning (ML) techniques to make more accurate predictions [8], [18], [26], [29], [39] than traditional methods. Although ML-based ways have performed far better than database management systems (DBMSs) on several publicly available datasets, many limitations remain.

Limitation I: Poor Robustness. Within-database models (WDMs) [18], [26], [39] train and test with different workloads on the same database for higher accuracy. However, the ability of WDMs to make accurate predictions is highly dependent on the training workload (Out of Distribution, OOD) [14], [21]. Unfortunately, choosing appropriate training queries for WDMs is difficult in practice [14]. Some works improve the model's accuracy on the new workload by retraining [25], [27]. However, when to retrain and how to collect the data used for retraining (also known as the cold start problem) are challenging. Besides, scenarios such as data drifts are prevalent in real applications [21]. Therefore, the poor robustness of WDMs leads to possible security issues when applied to DBMSs (may produce sub-optimal execution plans or incorrect scheduling).

Limitation II: Inadequate Accuracy. In recent years, across-database models (ADMs) have received attention from many researchers [8], [34]. Specifically, ADMs train and test on different databases and workloads (i.e., testing performed on workloads with unseen databases) to improve generalization. Compared to WDMs, ADMs have more robust generalization capabilities. However, ADMs may be unable to optimize the individual database as effectively as WDMs (also known as instance optimization [25]). As a result, the accuracy of ADMs cannot meet the requirements of commercial DBMSs [25].

Limitation III: Low Efficiency. Regardless of robustness and accuracy, the efficiency of existing cost estimation models (both training efficiency and inference efficiency) makes it challenging to meet the needs of commercial databases. For inference efficiency, either embedding into DBMS for better execution plan selection or performing query performance prediction (QPP) for resource scheduling. Both require cost estimation models with high inference speed. For training efficiency, faster-trained models can retrain or fine-tune more easily. Thus, they can adapt to more complex and changing scenarios. However, most ML-based cost estimation models sacrifice efficiency to achieve higher accuracy. In particular, ADMs, which build complex models to learn diverse databases, result in lower efficiency than WDMs. This paper argues that a superior cost estimation model must focus more than just accuracy and give up efficiency.

Among existing cost estimation models, including WDMs and ADMs, none meet the requirements of commercial systems. Since ADMs possess strong robustness. In addition, we contend that ADMs can be more effective and efficient. Therefore, in this paper, we focus on building an innovative ADM that simultaneously solves *limitations I*, *II*, and *III*, based on the following insights:

Insight I: The Struggle with Cardinality Estimation. Cardinality estimation predicts the number of rows returned in the query result, and its the basis for cost estimation[1]. However,

*The corresponding author, Kai Zheng, is with Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Shenzhen, China.

[1]The cost of a filter (e.g., t.production_year>2010) will change as the size or data distribution of the production_year column changes. For a join (e.g., t.id=mk.movie_id), when the sizes of tables t and mk vary or the result of a join is modified, the execution time will be affected. Therefore, data characteristics influence the execution time in the physical query plan via underlying cardinality data.

current solutions for cardinality estimation either lack robustness [4], [12] or are constrained to within-dataset estimations with inefficient inference [9], [35], [36]. Thus, leveraging cardinality estimation models directly for cost prediction is impractical. We contend that ADMs should avoid deriving information from data characteristics like filters or joins. While many cost estimation models prioritize learning these data characteristics, the inherent challenges in cardinality estimation might render this information potentially detrimental (*limitations I* and *II*) or the model inefficient (*limitation III*).

Insight II: The Power of Cost Correction. Correcting the estimated cost of DBMSs is a more efficient approach than learning data characteristics [33]. It sidesteps the complexity of cardinality estimation and instead learns the *error distribution of the query optimizer's estimated cost* (EDQO). In recent years, many studies have demonstrated the effectiveness of this approach on within-dataset [1], [17], [25]. Further, we find that the EDQO of different datasets is more transferable than the data characteristics. Therefore, we correct the research direction of cost estimation, especially ADMs, that learning the EDQO is a more effective and efficient way until the cardinality estimation task is properly addressed.

Insight III: Advancing cost estimation with pre-training. In recent years, pre-trained models have had an increasing number of applications in computer vision and natural language processing [5], [6], [24]. Pre-training has the following advantages [24]: (1) provides good model initialization, (2) avoids overfitting on small datasets, and (3) can learn a large amount of generalized knowledge to help downstream tasks. In cost estimation, the pre-trained model has two roles. (1) As a pre-trained estimator, the model can make accurate predictions directly on new scenarios or quickly adapt to unseen complex scenarios with only fine-tuning. (2) As a pre-trained encoder: The pre-trained generalized knowledge can help other cost estimation models improve their accuracy and robustness and solve the notorious cold-start problem (helping them to solve *limitations I* and *II*).

Based on the above observations, we establish a Database-Agnostic Cost Estimator named *DACE*, which promises accuracy, efficiency, and robustness in estimating costs across databases. *DACE* achieves efficient and robust inference (solves *limitations I* and *III*) without relying on the data characteristics of any specific dataset (*insight I*), instead focusing on learning the EDQO (*insight II*). It employs a lightweight transformer model to accomplish this, facilitating the parallel prediction of sub-plans through tree-structured attention. *DACE* designs with a tree structure-based loss adjustment strategy (solves *limitation II*). This design explicitly addresses the information redundancy observed in sub-plans. Finally, we extract generalized knowledge from *DACE* to improve existing WDMs (*insight III*).

Our contributions are summarized as follows:

- We discuss the relationship between cardinality and cost estimation, arguing that learning the EDQO is a more sensible way of cost estimation.
- We develop *DACE*, a novel cost estimation model designed for efficiency, accuracy, and robustness.
- We incorporate tree-structured attention and a tailored tree-structure-based loss adjustment strategy to facilitate sub-plans learning parallelly.
- To the best of our knowledge, this is the first time that knowledge of ADMs is utilized to enhance the performance of WDMs, effectively solving the cold start and OOD problems WDMs face.
- We extend the *across-database concept* to apply *DACE* to different machines with other performances.
- We conduct extensive experiments on several public datasets to demonstrate the effectiveness of *DACE*. Furthermore, we make our code publicly available[2].

We organize the remainder of the paper as follows. Sec. II introduces the problem. The *DACE* framework is detailed in Sec. III. Sec. IV presents the methodology of *DACE*. We show and analyze the experimental results in Sec. V. Sec. VI surveys related work, and Sec. VII concludes the paper.

II. Problem Definition

Before defining the problem, we first introduce the scenarios of cost estimation targeted in this paper, i.e., various forms of drifts, as shown in Fig. 1. (1) *Drift I*: Similar templates. The workload for training and testing is divided based on similar templates. The main drift is the restricted range of filters, and new join conditions rarely appear. (2) *Drift II*: New schema. Queries with unseen columns, tables, or join conditions appear, assuming the database is static. (3) *Drift III*: Data drift. Data updating (adding, deleting, or modifying data) and adding new columns or tables. This condition often accompanies Drift II. Some possible solutions are to recode the new features or to collect new data to retrain the model [14]. (4) *Drift IV*: Across-database. The test workload of the model appears on an unseen database. It is an extreme case of Drift III. (5) *Drift V*: Across-more. Drift I to IV assume the same DBMS and hardware setup (i.e., the training and testing workloads collect labels on the same machine). Across-more, on the other hand, breaks this assumption, and as a result, there exist much more complex scenarios than Drift IV.

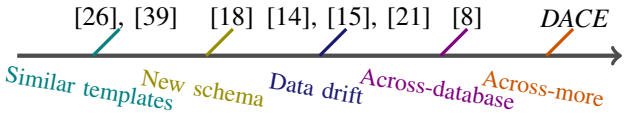

Fig. 1. Comparing model against data drifts. The horizontal axis from left to right indicates that the drifts are becoming more complex, and the model cannot handle the scenes to its right. For example, QPPNet [18] can take the new schema but cannot cope with data drift or more complicated tasks.

As a cost estimator, *DACE* has the same goal as previous works, namely to minimize $qerror$ (also known as Q-Error) [20], [26], [39], defined as follows:

$$qerror = \frac{max(est\_cost, cost)}{min(est\_cost, cost)}, \tag{1}$$

[2]https://github.com/liang-zibo/DACE

where est_cost represents the estimated execution time of the model. The $cost$ means the actual execution time, also known as latency. However, the goal of *DACE* is not only to achieve a minor $qerror$ than previous work, but we also propose the following purposes:

- Achieve across-dataset cost estimation with accuracy (i.e., a minor $qerror$) comparable to WDMs.
- *DACE* is a lightweight model with efficiency compared to existing ADMs and WDMs.
- *DACE* as a pre-trained model can effectively improve the accuracy and robustness of WDMs with little additional burden.

The evaluation for the above objectives is complex, and we will show the details in Sec. V.

III. Framework Overview

In this section, we introduce the framework of *DACE*. The workflow of *DACE* is shown in Fig. 2, where we first introduce data collection and feature extraction. After that, we introduce the *DACE* model. Finally, we present the application of *DACE* as a pre-trained model.

Data collection. Like previous works [8], [39], we collect the query plans corresponding to the query statements for training and inference. However, unlike WDMs [18], [26], [39], we create several databases and organize the corresponding workloads (i.e., query statements). Specifically, each plan[3] has a connected database [8].

Feature extraction. This part performs feature extraction on the query plan. First, we obtain the sequence of nodes corresponding to the query plan tree by Depth-First Search (DFS), which also allows us to get the tree-structured attention and the height of the nodes. After that, we encode each node by the encoder. In the training phase, we compute the loss weight for each node. This weight is determined based on the height of the nodes. A loss adjuster is used for this purpose and subsequent model updates.

***DACE* model.** *DACE* builds a lightweight estimation model. First, we apply a self-attention mechanism [30] to the plan encodings output from feature extraction and add the tree-structured attention for mask out. It ensures that DFS does not corrupt the tree structure information of the query plan tree. After that, we utilize a feed-forward neural network to obtain the hidden layer encoding. Using the predictor of a multilayer perceptron (MLP), we then predict the cost of all sub-plans in parallel. Finally, we utilize the dynamic loss weights output from the loss adjuster to compute the loss for updating the model. The learning of sub-plans is enhanced and prevents side effects caused by information redundancy.

Pre-trained model. As a pre-trained model, *DACE* serves two purposes:

1) As a pre-trained estimator. *DACE* enables accurate cost estimation in across-database scenarios and does not require fine-tuning parameters. However, relying only on pre-trained models for inference is hard in the across-more design. Therefore, we adopt a lightweight fine-tuning method, Low-Rank Adaptation (LoRA) [10]. It can adapt *DACE* to span more scenarios with only a tiny fine-tuning cost.

2) As a pre-trained encoder. *DACE* has the potential to provide a priori knowledge (also known as context [19]) for all WDMs through knowledge integration. Specifically, *DACE* acts as an encoder to learn the information embedded in the query plan. After that, we incorporate this information into a WDM encoder. *DACE* can improve the accuracy and robustness of WDMs. Moreover, due to the lightweight advantage of *DACE*, this knowledge integration imposes only a small additional burden on WDMs.

[3]In this paper, we use the terms "plan", "query plan" and "query plan tree" to represent the execution plan of the DBMS.

IV. Methodology

A. Data Collection

Our methodology for data collection follows Zero-Shot [8]. Specifically, we utilize a training workload Q_{train}, which comprises a series of query statements associated with a specific database D_{train} within the DBMS. For each query in Q_{train}, we procure the corresponding query plan, referred to as query plans P_{train}. These plans are generated by the DBMS's query optimizer (e.g., by EXPLAIN command in PostgreSQL), which devises an efficient execution strategy and includes the estimated information (i.e., cardinality and cost). It is worth noting that the test workload Q_{test} and its respective database D_{test} remain undisclosed to *DACE*. Nevertheless, for WDMs, subsets of Q_{test} or their associated plans P_{test} (same collection process as Q_{train}) may be utilized as training data.

B. Feature Extraction

This section extracts information from the query plan for *DACE* training or inference. Specifically, feature extraction includes an information catcher and an encoder. The information catcher uses DFS to obtain node sequences, adjacency matrices, and node heights. The encoder consists of a one-hot encoder, scaler, and loss adjuster and finally outputs the plan encoding.

Information catcher. As shown in Fig. 3, the information catcher first obtains the sequence of nodes corresponding to the query plan tree through DFS, after which it extracts node features, tree structure, and sub-plans information.

(1) Node features. The query plan tree provided by the DBMS consists of several nodes, and we obtain the corresponding sequence of nodes through DFS. Each node includes rich information, and we extract the features as follows: a) node type, b) DBMS-estimated cardinality, and c) DBMS-estimated cost. The node type indicates this node's specific operation type (seqscan, hashjoin, etc.). The DBMS calculates the cost based on the DBMS-estimated cardinality and the constant value of the cost model [13] (cpu_operator_cost, etc.). QPPNet [18] also employs this extraction method of node features. However, unlike QPPNet, we also extract tree structure and sub-plans information.

(2) Tree structure. The tree structure in a query plan tree implies the order of execution between nodes, i.e., the

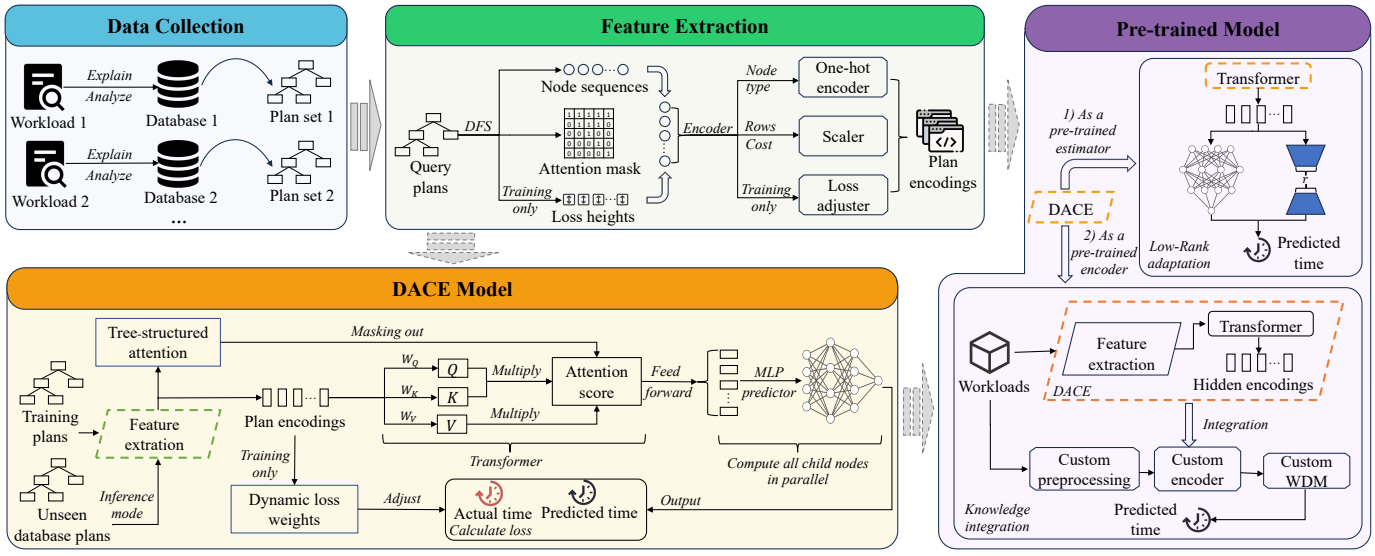

Fig. 2. DACE Framework Overview.

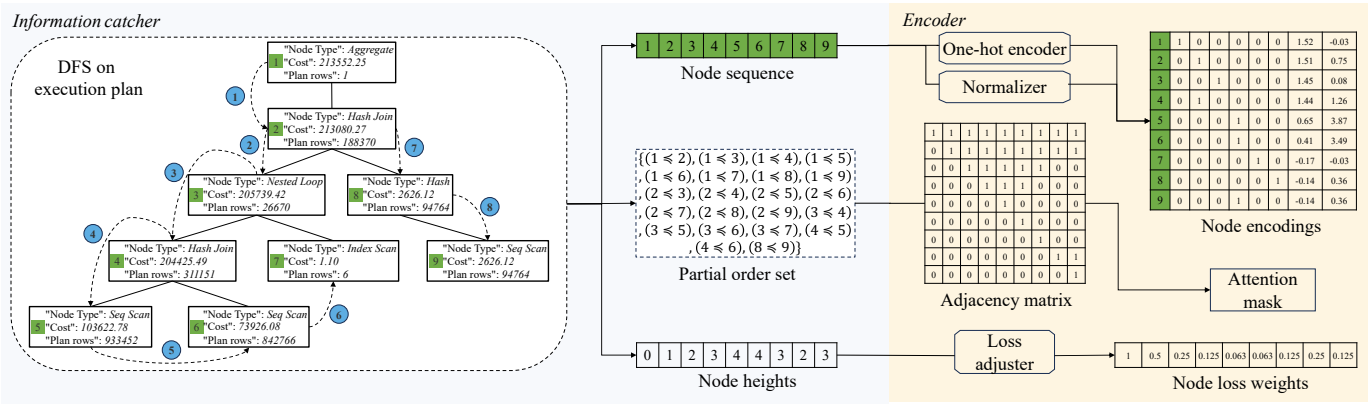

Fig. 3. An example of information catcher and encoder.

execution of a parent node follows that of its children. Inspired by QueryFormer [39], we define the query plan tree as a directed graph as follows:

$$G(p) = \{N_p, O_p\}, \tag{2}$$

where p is a query plan in P, and $G(p)$ denotes the directed graph corresponding to p, including the node sequence for the query plan N_p obtained by DFS, and the partial order set O_p [32]. Specifically, we define $node_i \leqslant node_j$ to mean that $node_i$ is the parent of $node_j$ ($node_i$ and $node_j$ are two different nodes in N_p). In addition, O_p has the following properties: (a) Reflexivity: $\forall node_i \in N_p, node_i \leqslant node_i$. (b) Antisymmetry: $\forall node_i, node_j \in N_p$, if $node_i \leqslant node_j$ and $node_j \leqslant node_i$ then $node_i = node_j$. (c) Transitivity: $\forall node_i, node_j, node_k \in N_p$, if $node_i \leqslant node_j$ and $node_j \leqslant node_k$ then $node_i \leqslant node_k$.

Based on $G(p)$, we can obtain the adjacency matrix $A(p)$ corresponding to p, defined as follows:

$$A(p)_{i,j} = \begin{cases} 1, & \text{if } node_i \leqslant node_j \\ 0, & \text{else} \end{cases} \tag{3}$$

Where $node_i$ is the ith node in the sequence of nodes acquired by DFS. In addition, O_p can be received during the DFS process, thereby obtaining $A(p)$ according to (3). Thus, acquiring $A(p)$ does not add an extra burden to feature extraction. In Sec. IV-C, we will utilize $A(p)$ to learn the tree structure, which ensures that DFS does not corrupt the tree-structured information of the query plan.

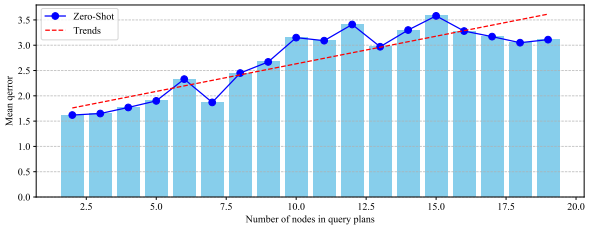

Fig. 4. Zero-Shot performance on query plans with different number of nodes. We use Zero-Shot's experimental setup to train on 19 databases at a time and test on the remaining one database for a total of 20 experiments. We report the average of the results of the 20 experiments.

(3) Sub-plans. Most of the existing cost estimation methods update the parameters of the model based only on the execu-

tion time of the root node (i.e., the execution time of the query plan). The drawback of this approach is that it fails to utilize the sub-plans information (i.e., the execution time of the sub-plans) in the query plan. It leads to difficulties in adapting the model to complex query plans. As shown in Fig. 4, the mean $qerror$ of Zero-Shot increases as the number of nodes increases. It indicates that query plans with a more significant number of nodes have more incredible learning difficulty. QPPNet [18] proposes a tree-structured model that can predict the cost of sub-plans and calculate the loss. However, its parent nodes need to wait for the outputs of the child nodes, resulting in slower inference. In addition, we find that QPPNet performs a large amount of repetitive learning, which we call *information redundancy*. The large number of sub-plans covers the information of the query plan, leading to less accurate estimations. Therefore, despite the importance of learning sub-plans for cost estimation, the following two issues need to be addressed: a) slow inference (discussed in "loss adjuster" of the encoder) and b) information redundancy (addressed in this section).

We design a tailored tree-structure-based loss adjustment strategy to handle information redundancy. Specifically, we make nodes with greater heights have lower loss weights, i.e., the loss on the node multiplies some weight between 0 and 1. The weight is calculated based on the height of the node. We define the height of a node as *the length of the shortest path from the node to its root node*. Therefore, in this section, we need to collect the height H_p of each node in p (we present the computation of weights in *loss adjuster*). Like A_p, we can also collect H_p during the DFS.

Encoder. As shown in Fig. 3, the encoder processes the results of the information catcher, which includes the one-hot encoder, scaler, and loss adjuster. Specifically, the node type is processed using the one-hot encoder, while the scaler addresses the DBMS information within the node features. Among them, DBMS information includes the cost and cardinality of DBMS estimation. The corresponding adjacency matrix $A(p)$ of $G(p)$ is obtained based on (3), which is the attention mask of the transformer. We design the loss adjuster in the training phase to realize the tailored tree-structure-based loss adjustment strategy. The loss weights of the nodes are calculated based on the nodes' height, which is the basis for updating the model.

(1) one-hot encoder. As in previous works [26], we use one-hot encoding to represent node types. As shown in Fig. 3, columns 2 to 7 of "Node encodings" represent one-hot encodings of node types. For the presentation, we only design six node types encodings in Fig. 3. However, we consider 16 node types in actual experiments (details in Sec. V).

(2) Scaler. Motivated by Zero-Shot [8], we use the robust scaler to process the cost and cardinality estimated by DBMS. As shown in Fig. 3, columns 8-9 of "Node encodings" represent the processed encodings.

(3) Loss adjuster. First, we explain why QPPNet's learning of sub-plans is flawed. The main reason is that learning the parent node also takes into account the information of the child nodes, which in turn leads to repeated learning of the child nodes. We call it *information redundancy*. As shown in Fig. 3, QPPNet learns node 5 (nodes 1 , 2 , 3 , and 4 are the parents of node 5) five times while learning node 1 only once.

To solve information redundancy, we design a loss adjuster based on the tailored tree-structure-based loss adjustment strategy. Specifically, the loss adjuster calculates the loss weights of the nodes through the node heights collected by the information catcher, defined as follows:

$$L_p = \alpha^{H_p}, \tag{4}$$

where L_p denotes loss weights corresponding to nodes in the query plan p, and we introduce α as a hyperparameter. As shown in Fig. 3, when $\alpha = 0.5$, the loss weights of nodes with heights from 0 to 4 are 1, 0.5, 0.25, 0.125, and 0.0625, respectively. Thus, *DACE* considers the learning of sub-plans and can utilize more knowledge of the query plan than previous works [8], [26], [39]. Compared to QPPNet, the tailored tree-structure-based loss adjustment strategy prevents repeated learning of sub-nodes by making them possess smaller loss weights.

C. DACE Model

In this section, we introduce the *DACE* model. Because QueryFormer [39] demonstrates the details of applying the Transformer to cost estimation. In this paper, we will not present the foundations of the Transformer. Instead, we introduce the key components that make the *DACE* model superior. These include (a) tree-structured attention, (b) parallel learning sub-plans, and (c) pre-trained model.

Tree-structured attention. Motivated by QueryFormer, we extract the tree-structured information in the query plan to apply to Transformer's attention mechanism. Specifically, we use the tree-structured attention obtained by the encoder as the attention mask of the Transformer, defined as follows:

$$\begin{cases} Q = SW_Q, K = SW_K, V = VW_V, \\ Attention(Q, K, V) = softmax(\frac{QK^T \odot M}{\sqrt{d}})V, \end{cases} \tag{5}$$

where $S \in \mathbb{R}^{n \times d}$ is the node encoding sequence output by the encoder, n is the number of nodes, and d is the length of the node encoding. $W_Q \in \mathbb{R}^{d \times d_k}$, $W_K \in \mathbb{R}^{d \times d_k}$ and $W_V \in \mathbb{R}^{d \times d_v}$ denote the parameter matrices of queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$, and values $V \in \mathbb{R}^{n \times d_v}$, respectively. And d_k is the dimensions of the queries and keys, and d_v is the dimensions of the values. $M \in \mathbb{R}^{n \times n}$ serves as the attention mask, derived from the adjacency matrix A. In this mask, we set 0 to a negative infinity value while keeping 1 unchanged.

Unlike QueryFormer, we do not introduce a learnable scalar (b_d in QueryFormer). Although b_d can dynamically measure the distance between nodes, a simple scalar must be improved to represent complex query plans. Instead, we devise a parallel learning approach for sub-plans. It effectively and efficiently utilizes all the sub-plans information in the query plan.

Parallel learning sub-plans. In Sec. IV-B, we introduce the motivation for learning sub-plans. Most existing work focuses only on the loss of the root node to update the model. Although QPPNet can predict the cost of the sub-plans and obtain the loss, its parent node needs to wait for the computation results of all the child nodes before making a prediction. We firmly believe that the inference speed of the cost estimation model is crucial [25]. Therefore, we utilize the structure of Transformer and the tree-structured attention mask M to enable parallel prediction of sub-plans, defined as follows:

$$est\_cost(sub_p) = MLP(Attention(Q, K, V)), \quad (6)$$

where $Attention(Q, K, V) \in \mathbb{R}^{n \times d_v}$ is obtained via (5), sub_p denotes all the sub-plans of p. MLP consists of a three-layer fully connected linear network (details in Sec. V). Ultimately, we obtain the estimations of sub_p costs $cost(sub_p)$. The *DACE* model enables parallel learning through two components: (a) Transformer and (b) MLP. In the Transformer, the model concurrently computes the attention of all sub-nodes, resulting in the hidden state (denoted as $Attention(Q, K, V)$) for each node. It is worth noting that by employing tree-structured attention, each node computes based only on the information of its children. It is the same logic as the actual execution of the query plan (i.e., the execution of the parent node must come after the child nodes). Therefore, although we achieve efficient parallel prediction of subplans, we also consider tree-structured information for accurate prediction compared to QPPNet. In MLP, the model can predict the costs of sub_p based on $Attention(Q, K, V)$ in parallel. Therefore, *DACE* can efficiently obtain the estimated costs of all sub-plans of a query plan through parallel computation.

In Sec. IV-B, we contend that QPPNet's learning of sub-plans leads to information redundancy. Therefore, we design the loss adjuster based on the tailored tree-structure-based loss adjustment strategy. Its role is to set lower loss weights for nodes with greater height (motivation and details in Sec. IV-B). Based on the loss adjuster, we define the training loss as follows:

$$loss_p = \sum_{i=1}^{n} (L_{p,i} \cdot qerror(est\_cost(sub_p)_i, cost(sub_p)_i)) \quad (7)$$

$L_{p,i}$ denotes the loss weight of the ith node (DFS determines the order) of L_p. $est\_cost(sub_p)_i$ and $cost(sub_p)_i$ represent the estimated and actual cost of the ith node of sub_p, respectively. Ultimately, we obtain the loss $loss_p$ of the query plan p. To clearly show the effect of node weights on loss, we use summation to compute $loss_p$. However, we implement this process in a parallel way. There is no additional burden on the training of the model. In Sec. I, we discuss the motivation for estimating the parameters of *DACE* using MLE. Therefore, we perform backpropagation based on $loss_p$ to solve for the optimal parameters $\widehat{\theta}$ of *DACE*.

D. Pre-trained Model

This section describes two uses of *DACE* as a pre-trained model, i.e., a pre-trained estimator or a pre-trained encoder.

Pre-trained estimator. In most scenarios, *DACE* can make accurate predictions directly without fine-tuning (e.g., Zero-Shot's few-shot approach). However, we further extend the concept of across-database by proposing across-more. In this paper, we define across-more as testing on unseen databases on hardware devices with different configurations. Specifically, we obtain workloads Q^1_{train} and Q^1_{test} on machine $M1$ (*DACE* is trained on Q^1_{train} to get $\widehat{\theta}$). The same query statements are then executed on machine $M2$ to fetch new workloads Q^2_{train} and Q^2_{test}. It is hard to test directly on Q^2_{test} based on $M2$ since different machines may have the different EDQO. However, we argue that the knowledge possessed by *DACE* is migratable, i.e., it can quickly adapt to across-more. Unlike Zero-Shot, we use Low-Rank Adaptation (LoRA) [10] to fine-tune *DACE*, defined as follows:

$$\begin{cases} h_i = h_{i-1} W_i + h_{i-1} \triangle W_i, \\ \triangle W_i = W_{B_i} W_{A_i} \end{cases} \quad (8)$$

$W_i \in \mathbb{R}^{d_{i-1}}$ ($i = 1, 2, 3$ and $d_0 = d_v$) denotes the weights of the ith network layer of the MLP, and $h_i \in \mathbb{R}^{n \times d_i}$ represents the hidden layer of the output of the ith network layer. $\triangle W_i \in \mathbb{R}^{d_i}$ is the parameters of LoRA, which consists of $W_{B_i} \in \mathbb{R}^{d_{i-1} \times r_i}$ and $W_{A_i} \in \mathbb{R}^{r_i \times d_i}$ and has $r_i \ll min(d_{i-1}, d_i)$. During the training phase, we updated only W_i and set $\triangle W_i$ untrainable. Conversely, during fine-tuning, we update only $\triangle W_i$ and select W_i to be untrainable. Since $r_i \ll min(d_{i-1}, d_i)$, the training cost of $\triangle W_i$ is less than that of W_i. Specifically, *DACE* fine-tuning is 192% more efficient than retraining (details in Sec. V).

Pre-trained encoder. Knowledge integration (transfer learning) has many applications in computer vision and natural language processing [22], [41]. It effectively enhances the performance of task-specific models by offering generalized contexts across various tasks [19]. In cost estimation, *DACE*, as a superior ADM, can give context for WDMs. Specifically, *DACE* can seamlessly integrate with WDMs as a pre-trained encoder. We take MSCN as an example and define it as follows:

$$w_{out} = MLP_{out}([w_T, w_J, w_P, w_E]), \quad (9)$$

where MSCN [12] defines w_T, w_J, and w_P and denote the table, join, and predicate information of the query, respectively. w_E is the output of *DACE* as a pre-trained encoder and has $w_E = h_2$ (based on (8)).

V. Experiments

A. Experimemt Setting

Datasets. For a fair comparison, we use the benchmark proposed by Zero-Shot. It consists of 20 different databases (including IMDB, TPC-H, etc.), and there is a large gap between them regarding schemas, tables, columns, etc.

Workloads. (1) Workload 1: Following Zero-Shot, we generated 10000 query statements for each database (complex queries in Zero-Shot). We select queries from each database for testing, and queries from the remaining 19 databases,

except the test database, are used for training. We execute all the query statements on $M1$ to obtain the execution time and query plan[4] (each query statement corresponds to a query plan). (2) Workload 2: In this paper, we propose the across-more scenario. Specifically, we take the query statements of workload 1 and execute them on $M2$ to obtain new labels and query plans. The training and testing of workload 2 is the same as that of workload 1. (3) Workload 3: We adopt the benchmark proposed by MSCN [12] based on IMDB as workload 3. Its training set includes 100000 query statements. The test set comprises synthetic, scale, and job-light with 5000, 500, and 70 queries, respectively. Like workload 1, we obtain the execution times and query plans based on $M1$. It is worth noting that WDMs need to be trained on the training set of workload 3 first and tested afterward. *DACE*, on the other hand, can test directly, i.e., without utilizing any IMDB queries for training. Combining the above workloads, we answer three crucial questions: (a) why is *DACE* accurate? (*limitation I*), (b) why is *DACE* efficient? (*limitation II*) and (c) why is *DACE* robust? (*limitation III*)

Baselines. We compare *DACE* with representative cost estimation models, including ADMs and WDMs. (1) ADMs. **Zero-Shot [8]:** Zero-Shot transforms query plans into directed graphs. Build node-type-specific MLPs to learn different node information. It is reasoning through bottom-up information propagation to predict execution time. (2) WDMs. **MSCN [12]:** MSCN is a learning-based cardinality estimation model. It extracts tables, joins, and predicates information from query statements into a Deep Set for prediction. **QPP-Net [18]:** QPPNet accepts the query plan tree as an input and predicts its execution time. QPPNet learns the nodes in the query plan by different networks. The outputs of its children nodes are used as inputs to the parent nodes to predict the execution time. **TPool [26]:** TPool extracts diverse information from the query plan, including tables, joins, and predicates. It also learns string predicates through string embedding. Finally, it predicts both the cardinality and execution time of the query plan through multi-task learning. **QueryFormer [39]:** QueryFormer is designed with height encoding, tree-bias attention, and Super node and learns the query plan through an encoder with eight layers of Transformer. QueryFormer can be applied to a variety of tasks in database optimization.

Implementation details. The implementation details of our experiments are as follows:

Parameters Setting. We consider a total of 16 node types, so the encoding length of each node of the feature extraction output is 18 (including the cost and cardinality estimated by DBMS). $d = 18$ and $d_k = d_v = 128$ in Transformer. The value of α in the loss adjuster is 0.5 by binary search. Furthermore, *DACE* consists of only one encoder layer and does not utilize the multi-head attention mechanism. W_1, W_2 and W_3 in the MLP of *DACE* are set to 128, 64, and 1, respectively. r_1, r_2 and r_3 in LoRA are set to 32, 16, and 8, respectively.

Hardware and Platform. The hardware devices and platforms used in our experiments are as follows: machine $M1$: the CPU is Intel(R) Xeon(R) CPU E5-2650 v4@2.20GHz, and the GPU is GeForce GTX 1080 Ti. Machine $M2$: the CPU is Intel(R) Core(TM) CPU i5-8500@3.00GHz. The DBMS selected is PostgreSQL14.5, the algorithms are implemented based on Python 3.9.18. Other software versions can be obtained from our code[5].

B. DACE Accuracy

As a pre-trained estimator, *DACE* can make accurate predictions. As a pre-trained encoder, *DACE* can provide WDMs with information-rich encoding, thus helping them to make more accurate predictions. To evaluate the accuracy of *DACE*, we conduct extensive experiments on workloads 1, 2, and 3. We also compare *DACE* with currently representative ADMs and WDMs. First, we compare *DACE* with ADMs. We design an experimental scenario similar to Zero-Shot (workload 1) and propose a more complex across-more scenario (workload 2). Then, we argue that *DACE* can outperform WDMs even when tested directly without knowledge of the specific database (workload 3). Finally, we use *DACE* as a pre-trained encoder to provide query plan embeddings for WDMs. Due to the superiority of *DACE*, this simple knowledge integration method can effectively improve the accuracy of all WDMs.

Comparison with ADMs. We follow the experimental setup of Zero-Shot. We trained each model on 19/20 databases and later tested it on the remaining database. For PostgreSQL, the estimated cost is not in the same units as the execution time, so we processed it with a linear model as the execution time predicted by PostgreSQL. Fig. 5 shows the overall accuracy of *DACE* on workloads 1 and 2. The median $qerror$ of *DACE* is less than Zero-Shot on 16 of the databases. In addition, the median $qerror$ of *DACE* is less than 1.48 on all of the databases (compared to 1.56 for Zero-Shot).

In addition, we conducted experiments on the adaptability of the *DACE* model in across-more scenarios. We first train the *DACE* model on workload 1 and apply the LoRA technique [10] to fine-tune it on workload 2, thus obtaining the optimized *DACE*-LoRA model. For example, we select the IMDB database as the test set for workload 1. In that case, we trained the *DACE* model on 19 other databases except IMDB and evaluated it on IMDB. After completing this phase, we fine-tuned the *DACE* model on the 19 databases of workload 2 (excluding IMDB) to obtain *DACE*-LoRA. Surprisingly, *DACE*-LoRA performs well on workload 2, exceeding the performance of *DACE* on workload 1. Specifically, the median qerror of *DACE*-LoRA on workload 2 are all lower than 1.27, a result that highlights that the *DACE* model can be effectively adapted to a broader range of scenarios through fine-tuning. In addition, we further reveal the significant value of general knowledge by analyzing the experiments on workload 2.

[4]We use the EXPLAIN ANALYZE command in PostgreSQL to execute and obtain the query plan corresponding to each query statement. It is worth noting that the features input to the model include only the information estimated by the query optimizer in the query plan (e.g., estimated cardinality and cost), and we use the actual execution time only as a label.

[5]https://github.com/liang-zibo/*DACE*

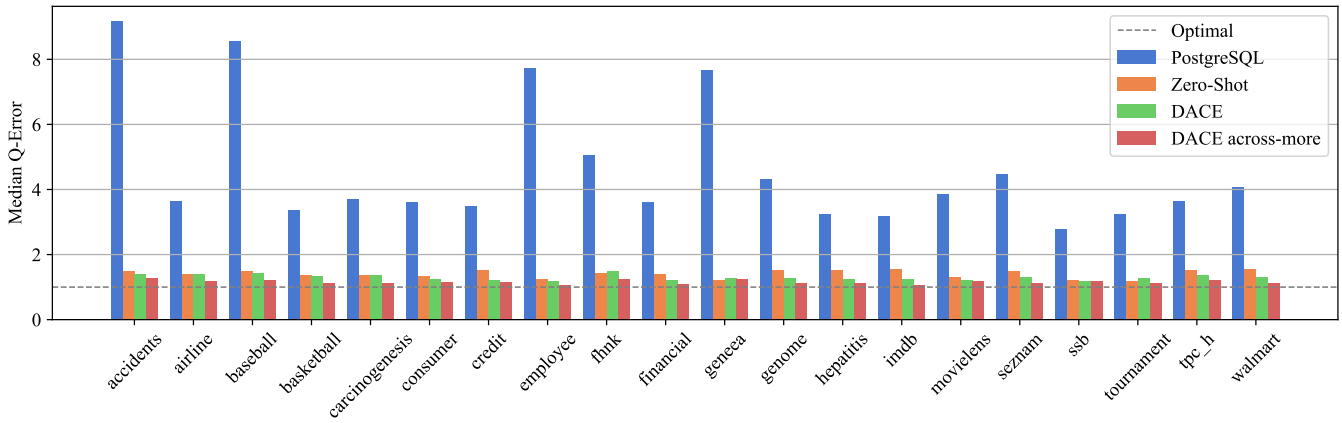

Fig. 5. *DACE*'s overall accuracy on workload 1 and workload 2. We trained *DACE* and Zero-Shot models on 19/20 databases and tested on the remaining database (workload 1). *DACE* across-more was further fine-tuned on 19/20 databases (based on LoRA) and similarly tested on the remaining databases (workload 2).

DACE-LoRA fuses the knowledge from workloads 1 and 2 and exhibits even better performance. This finding reinforces our view that learning EDQO, a general knowledge, is a much more effective way.

Comparison with WDMs. Although ADMs have better robustness, they may not be as accurate as WDMs (called instance optimization in Auto-WLM [25]) on the specific database. However, even on the specific database, *DACE* can outperform WDMs. Tab. I shows the performance of various cost estimation models on workload 3. Notably, *DACE* and Zero-Shot were trained on 19/20 databases (excluding IMDB) and did not utilize any information from IMDB. In contrast, we train WDMs on 100,000 queries based on IMDB. It can be seen that QueryFormer's median $qerror$ on all three test sets is lower than Zero-Shot's, and Zero-Shot does not significantly outperform QueryFormer on the remaining metrics. However, *DACE* outperforms all metrics (except for Synthetic's median $qerror$) on all three test sets (including WDMs and ADMs). Specifically, *DACE* outperforms the best existing models (Zero-Shot and QueryFormer) by more than $10\times$, $13\times$, and $23\times$ in terms of maximum $qerror$ on Synthetic, Scale, and JOB-light, respectively. Thus, we can conclude that *DACE* can make more accurate predictions than WDMs on a specific database and does not utilize any knowledge of that database for fine-tuning (i.e., *DACE* achieves instance optimization).

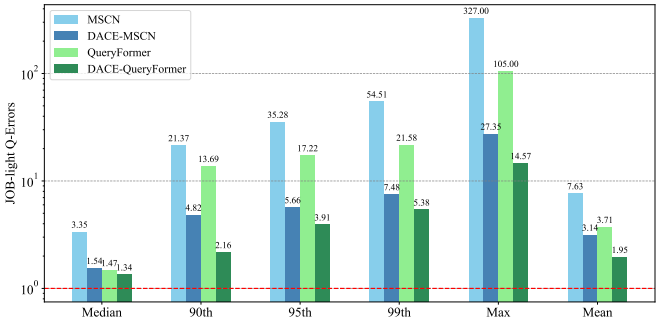

Fig. 6. Comparison of MSCN and QueryFormer with and without *DACE*.

As a pre-trained encoder. As a pre-trained encoder,

TABLE I
$qerror$ ANALYSIS FOR COST ESTIMATION ON WORKLOAD 3. *DACE* AND ZERO-SHOT MODELS UNTRAINED ON IMDB WORKLOADS.

Synthetic	**Median**	**90th**	**95th**	**99th**	**Max**	**Mean**
PostgreSQL	3.58	15.49	17.62	137.14	983	13.47
MSCN	2.19	4.95	12.37	47.52	376	5.34
QPPNet	1.33	3.39	8.52	37.15	103	3.49
TPool	1.45	3.57	8.24	23.63	219	3.51
QueryFormer	1.18	2.03	5.40	13.93	127	1.54
Zero-Shot	1.34	2.19	4.96	8.67	52.60	1.50
DACE	1.23	1.70	1.98	2.45	4.47	1.32
DACE-LoRA	**1.14**	**1.49**	**1.67**	**2.30**	**3.31**	**1.22**
Scale	**Median**	**90th**	**95th**	**99th**	**Max**	**Mean**
PostgreSQL	3.19	12.74	19.68	65.38	544	15.82
MSCN	2.06	5.74	11.34	20.83	204	6.17
QPPNet	1.30	3.95	9.64	13.64	84	3.58
TPool	1.53	4.06	10.37	15.47	192	4.06
QueryFormer	1.32	2.26	2.95	5.63	59.81	1.52
Zero-Shot	1.46	2.18	3.64	6.49	63.79	1.62
DACE	1.25	1.89	2.26	2.96	4.44	1.37
DACE-LoRA	**1.16**	**1.67**	**1.88**	**2.55**	**3.41**	**1.28**
JOB-light	**Median**	**90th**	**95th**	**99th**	**Max**	**Mean**
PostgreSQL	4.90	25.36	43.58	67.29	852	17.49
MSCN	3.35	21.37	35.28	54.51	327	7.63
QPPNet	2.94	17.58	19.37	23.41	48.01	4.72
TPool	3.28	23.58	38.54	45.82	254	6.72
QueryFormer	1.47	13.69	17.22	21.58	105	3.71
Zero-Shot	1.58	2.57	5.05	6.38	92.47	2.84
DACE	1.30	2.39	2.64	3.58	3.72	1.52
DACE-LoRA	**1.18**	**1.83**	**2.12**	**2.75**	**2.90**	**1.32**

DACE can provide all WDMs with embeddings of query plans through knowledge integration. WDMs can have better accuracy and robustness by utilizing the embeddings provided by *DACE*. Besides, they can solve the notorious cold start problem effectively. In this section, we demonstrate *DACE*'s accuracy improvements to WDMs (robustness and the cold start problem details in Sec. V-D).

We integrate *DACE* into MSCN (defined in (9)) and QueryFormer. And we obtain *DACE*-MSCN and *DACE*-QueryFormer by training on workload 3. Fig. 6 illustrates the performance of the models on JOB-light. The maximum $qerror$ of MSCN and QueryFormer are $11\times$ and $7\times$ higher than those of *DACE*-MSCN and *DACE*-QueryFormer, respectively. Therefore, as a pre-trained encoder, *DACE* can effectively improve the accuracy of WDMs.

Discussion. An important question is: why can only *DACE* adapt to across-more scenarios or act as an effective pre-trained encoder? Zero-Shot can also be fine-tuned with LoRA and achieve competitive results on across-more scenarios. Similarly, Zero-Shot can also provide query plan embeddings for knowledge integration. However, a superior cost estimation model should balance accuracy and efficiency (both training and inference efficiency). Zero-Shot's model size is $33\times$ larger than *DACE* (details in Sec. V-C). Thus, Zero-Shot is less efficient than *DACE* in training, fine-tuning, or inference, making *DACE* only less burdensome for fine-tuning or as a pre-trained encoder.

Furthermore, since *DACE* is efficient, it is more appropriate to adapt it to a specific database (or across-more scenarios) by fine-tuning (details in Sec. V-C). As shown in Tab. I, we obtain *DACE*-LoRA based on the training workload of workload 3 by updating only $\triangle W$ (defined in (8)). *DACE*-LoRA outperforms *DACE* in all metrics, and its accuracy is further improved. Therefore, the ability to make accurate predictions is only one of the advantages of *DACE*, and the lightweight nature of *DACE* can further expand its application scenarios. In Sec. V-C, we will analyze the efficiency of *DACE*.

Why is *DACE* accurate? Making accurate predictions is the essential task of cost estimation modeling. Although ADMs are more robust than WDMs. Their limitation is that they are less accurate than WDMs on a particular database. However, the experimental results in this section illustrate that *DACE* outperforms WDMs in accuracy. The main reason for *DACE*'s superior accuracy is that we design tree-structured attention and parallel learning of sub-plans to correct EDQO. In particular, we employ a tailored tree-structured loss-based adjustment strategy to equip sub-nodes with smaller loss weights to address the information redundancy problem. We argue that learning EDQO is a more efficient approach until the base estimation task is solved correctly. As a result, *DACE* still outperforms WDM, even on specific database workloads. Moreover, EDQO is a general knowledge with better transferability than predicate information. Therefore, *DACE* also has excellent accuracy in across-database scenarios.

C. DACE efficiency

We present the efficiency of *DACE* in three aspects: (1) model size, (2) training efficiency, and (3) inference efficiency. The above efficiency makes *DACE* easy to deploy. In addition, efficient training allows *DACE* to be fine-tuned with less cost, thus adapting to more complex environments. Finally, *DACE* has an inference efficiency that can meet the requirements of real application scenarios. Efficient inference makes *DACE* as a pre-trained encoder does not bring extra burden to WDMs. **Model size.** We report the number of parameters for each cost estimation model, as shown in Tab. II. Except for *DACE*, MSCN has the most miniature model. The model sizes of MSCN, QPPNet, TPool, QueryFormer, and Zero-Shot are $39\times$, $44\times$, $59\times$, $133\times$, and $42\times$ larger than *DACE*, respectively. We also show the model size of *DACE*-LoRA. It can be seen that *DACE*-LoRA is 25% larger than the model of *DACE*. However, MSCN, QPPNet, TPool, QueryFormer, and Zero-Shot model sizes are $31\times$, $35\times$, $47\times$, $106\times$, and $33\times$ larger than *DACE*-LoRA, respectively. Therefore, even if we introduce additional parameters to *DACE* based on LoRA to obtain *DACE*-LoRA, the number of model parameters of *DACE*-LoRA is still much smaller than the other models.

TABLE II
EFFICIENCY ANALYSIS OF THE MODELS

Model	Model size (MB)	Training efficiency (queries/sec)	Inference efficiency (queries/sec)
PostgreSQL	-	-	1035
MSCN	2.509	675	978
DACE-MSCN	2.611	633	824
QPPNet	2.836	436	592
TPool	3.808	549	697
QueryFormer	8.553	473	638
DACE-QueryFormer	8.714	455	609
Zero-Shot	2.708	307	426
DACE-LoRA	0.080	**40426 (tuning)**	29561
DACE	**0.064**	21031	**33995**

Training efficiency. We set the batch size for each model training to 512. Tab. II shows the training efficiency of each cost estimation model (based on workload 3). Specifically, the training efficiency of *DACE* is $31\times$, $48\times$, $38\times$, $44\times$, and $68\times$ higher than that of MSCN, QPPNet, TPool, QueryFormer, and Zero-Shot, respectively. In addition, the inference efficiency of *DACE*-LoRA is $1.92\times$ that of *DACE*. It is because *DACE*-LoRA only needs to update the parameters associated with LoRA (i.e., $\triangle W$ in (8)). Therefore, *DACE* has superior training efficiency. And *DACE*-LoRA has lower fine-tuning expenses.

Inference efficiency. Inference efficiency is an essential metric for evaluating cost estimation models. Tab. II demonstrates the inference efficiency of PostgreSQL and learning-

based cost estimation models[6]. The inference efficiency of all learned models is lower than that of PostgreSQL except *DACE*. And *DACE* is 32× more efficient than PostgreSQL. Therefore, based on *DACE*'s superior inference efficiency, *DACE* as a pre-trained encoder does not impose an additional burden on other models (as shown in Tab. II).

Why is *DACE* efficient? Auto-WLM [25] argues that inference efficiency is one of the main limitations of ADMs. However, we contend that a superior ADM (i.e., *DACE*) can also have efficient inference. As shown in Tab. II, *DACE* is superior to other baselines regarding model size, training efficiency, and inference efficiency. The reasons for this are: First, unlike previous work, *DACE* learns EDQO rather than data characteristics. It means that we have more straightforward coding. Later, since EDQO is easier to handle, we only need a lightweight transformer to make accurate predictions. Finally, we utilize the sub-plan information in the query plan to make the prediction more accurate. However, through parallel learning, *DACE* takes only a small additional burden. Therefore, *DACE* is more efficient than other baselines (including WDMs).

D. DACE Robustness

Fig. 5 demonstrates not only the accuracy but also the robustness of *DACE* (across-database in Fig. 1). In this section, we first show the performance of *DACE* on data drift scenarios. After that, we analyze the effect of the training workload's size on *DACE*. Finally, we offer that *DACE* as a pre-trained encoder can effectively improve the robustness of WDMs.

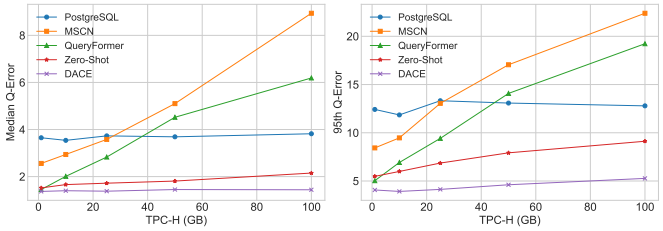

Fig. 7. Robustness of *DACE* on the data drift scenario.

Data drift. Data drift alters the execution time of query statements, challenging existing cost estimation models to maintain accuracy without retraining. And most models need to collect a new workload for retraining. As a superior ADM, *DACE* can adapt better to data drift scenarios. Fig. 7 shows the performance of *DACE* and other cost estimation models on the data drift scenario. We trained on 19/20 databases (excluding TPC-H) to obtain ADMs (i.e., *DACE* and Zero-Shot). Regarding WDMs (i.e., MSCN and QueryFormer), we generate 50,000 queries for them for training (based on TPC-H (1GB).) The test datasets for ADMs and WDMs are 10,000 queries, and we execute them to obtain the labels on different-sized TPC-H.

As shown in Fig. 7, *DACE* has the highest accuracy across the full range of test workloads. Specifically, the maximum degradation of the median and 95th $qerror$ during data drift is 5% and 29% for *DACE* (41% and 66% for Zero-Shot). Except for *DACE*, Zero-Shot has the best performance. It indicates that ADMs are more adept at handling data drift scenarios than WDMs. WDMs have higher $qerror$ than PostgreSQL when TPC-H is larger than 50GB. Therefore, most WDMs find it challenging to cope with data drift scenarios. On the other hand, *DACE* can still make accurate predictions even in data drift scenarios and does not need to fine-tune any parameters. It makes *DACE* more suitable for deployment in real applications.

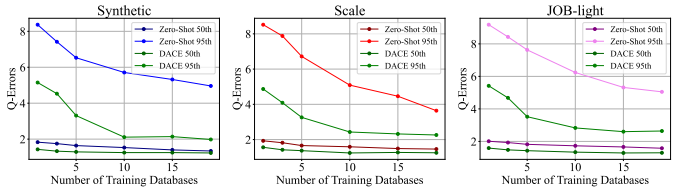

Fig. 8. *DACE* and Zero-Shot accuracy by the number of training databases.

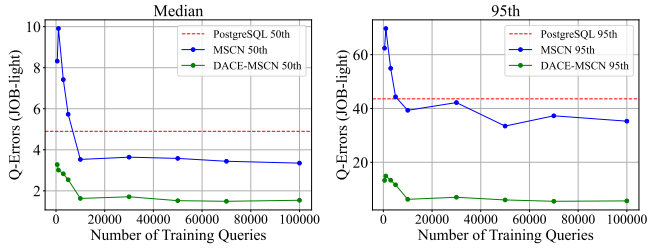

Fig. 9. Performance of MSCN with and without *DACE* by the number of training queries.

Training overhead. In most scenarios, as the training overhead (i.e., the number of training queries increases) increases, the model will perform better. However, a superior cost estimation model should only require less training overhead to achieve competitive performance. Essentially, this means that the model is more robust and better able to cope with the cold start problem. In this section, we show the correlation between the performance of *DACE* and the training overhead.

We gradually increase the number of databases used for training and test the model's performance. Specifically, we set the training databases to 1, 3, 5, 10, 15, and 19 (without IMDB) to test them on the same workload. Fig. 8 shows the performance of Zero-Shot and *DACE* on Synthetic, Scale, and JOB-light workloads. Zero-Shot needs about 10 to 15 training databases to achieve stable performance. *DACE*, on the other

[6]This efficiency comparison is unfair to PostgreSQL but enables a more straightforward comparison of the efficiency differences of ML-based models. Because, except MSCN, all other models (i.e., query-plan-based cost estimation models) require query plan output from PostgreSQL as input. Therefore, the runtime of a plan-based model consists of two aspects: (1) the time to obtain the query plan from PostgreSQL and (2) the time for the model to train or test using the query plan. However, to compare the efficiency of ML-based cost estimation models more clearly. We did not consider the effect of the time consumed by PostgreSQL on QPPNet, TPool, QueryFormer, Zero-Shot, and *DACE*.

hand, requires only 3 to 5 training databases for accurate prediction. Therefore, *DACE* has better performance with the same training overhead. In addition, considering the excellent efficiency of *DACE* (in Sec. V-C), the training cost of *DACE* is much smaller than existing cost estimation models in practical applications.

Enhancing the robustness of WDMs. As a pre-trained encoder, *DACE* can effectively improve the robustness of WDMs. Further, *DACE* can help WDMs to solve the notorious cold start problem. Fig. 9 shows the $qerror$ by the number of training queries for *DACE*-MSCN and MSCN. MSCN needs about 5,000 to 10,000 queries to achieve a performance comparable to PostgreSQL. *DACE*-MSCN, on the other hand, outperforms PostgreSQL on 100 to 100,000 training queries. In addition, the prediction accuracy of *DACE*-MSCN with only 100 training queries is still higher than that of MSCN (both median and 95th $qerror$). Therefore, as a pre-trained encoder, *DACE* can effectively improve the accuracy (in Sec. V-B) and robustness of WDMs. And it will not bring extra burden to WDMs (in Sec. V-C).

Why is *DACE* robust? In this section, we show the performance of *DACE* on the robustness test. From this, we can see that *DACE* has a more robust adaptability to scenarios such as data drift than previous methods. Furthermore, it demonstrates adaptability even in extreme drift scenarios (i.e., across-database and across-more), pushing the boundaries of WDM capabilities. The main reason why *DACE* is robust is that the EDQO is more appropriate than cardinality. We did not learn the cardinality information embedded in the data characteristics as in previous work [8], [12], [18], [26], [39]. Instead, we correct for the cost estimated by the DBMS, i.e., we learn the EDQO. In addition, *DACE* can learn the EDQO of different databases compared to WDMs. Therefore, in practical applications, *DACE* can utilize richer data than WDMs to make the model more robust.

E. Ablation Study

In this section, we analyze the impact of critical components of *DACE* on its performance. First, we show the effects of tree-structured attention on the performance and robustness of *DACE*. After that, we offer that the tree-structure-based loss adjustment strategy (i.e., loss adjuster) can effectively solve the information redundancy problem. Finally, we analyze the effectiveness of parallel learning sub-plans.

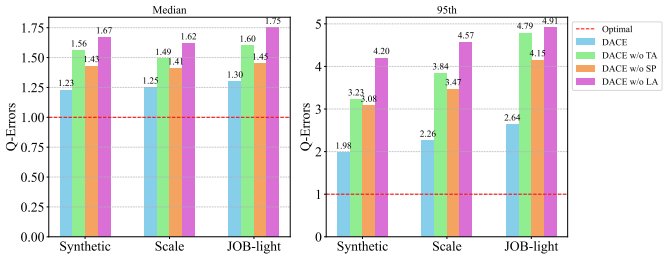

Fig. 10. Impact of tree-structured attention and sub-plans information on the performance of *DACE*. We train *DACE* on 19/20 databases (excluding IMDB).

Tree-structured attention. We introduce tree-structured attention to learn the tree-structured information of a query plan. Unlike QueryFormer, *DACE* does not use height embedding, a learnable scaler, or the super node. The main reason is that the height and learnable scaler information is embedded in tree-structured attention. In addition, *DACE* learns the EDQO rather than data characteristics. Hence, *DACE* enhances performance without the necessity of complex models. Moreover, we need to maintain the efficiency of *DACE*. Therefore, we only use tree-structured attention to represent tree structure information efficiently.

Fig. 10 illustrates the performance comparison between *DACE* and *DACE* without tree-structured attention (i.e., *DACE* w/o TA). Regarding median $qerror$ on the three test workloads, *DACE* leads *DACE* w/o TA by 21%,16% and 18%, respectively. Thus, tree-structured attention can effectively represent the tree-structured information of the query plan. In turn, it improves the accuracy and robustness of *DACE* (we test it on unseen databases).

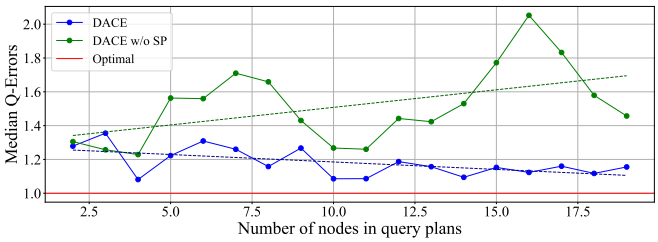

Fig. 11. *DACE* and *DACE* w/o SP performance on query plans with different number of nodes. We train the models on 19 databases, and tested on 10,000 query statements on IMDB.

Loss adjuster. We design the loss adjuster based on the tree-structure-based loss adjustment strategy. Here, we analyze the effect of the loss adjuster on the performance of *DACE*. Specifically, we analyze the value of α. When $\alpha = 0$ (i.e., *DACE* w/o SP), only the root node has a loss weight of 1, and the rest of the nodes are 0. At this time, *DACE* cannot learn any information about the subplans. When $\alpha = 1$ (i.e., *DACE* w/o LA), similar to QPPNet, all nodes have the same loss weight, implying that the loss adjuster fails. We set the value of α to 0.5 for *DACE* by binary search.

Fig. 10 illustrates the performance comparison of *DACE* ($\alpha = 0.5$), *DACE* w/o SP ($\alpha = 0$) and *DACE* w/o LA ($\alpha = 1$). It can be seen that *DACE* has the lowest $qerror$. While *DACE* w/o LA has the largest prediction errors. Therefore, learning the sub-plan information can effectively improve the model's performance. However, we must solve the resulting information redundancy problem (*DACE* solved by the loss adjuster).

We also show how well *DACE* handles complex query plans (i.e., with larger nodes). Fig. 11 shows the performance of *DACE* and *DACE* w/o LA on query plans with the different number of nodes. It can be seen that the $qerror$ of *DACE* w/o LA increases as the number of nodes increases. On the other hand, *DACE* receives almost no effect of the number of

nodes on its performance. Therefore, by learning the subplans, *DACE* can handle complex query plans effectively.

Most importantly, learning the sub-plans does not affect the efficiency of *DACE*. Specifically, learning all sub-plans during training costs about 10% to 15% extra burden. On the other hand, there is no additional overhead for the inference process (it only predicts the cost of the root node during inference). Therefore, the tree-structure-based loss adjustment strategy we designed is efficient and effective.

Comparison with Cardinality. Most existing cost estimation models incorporate predicate information in the encoding. In Sec. I, we argue that these methods learn the cardinality information embedded in the predicates. Unlike previous work, *DACE* corrects EDQO instead of learning cardinality information from scratch. In the last experimental sections, we argued that *DACE* is more accurate, efficient, and robust than existing baselines. We concluded that the main reason for this is that correcting EDQO has more advantages than learning cardinality.

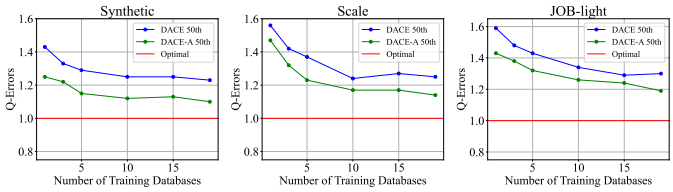

Fig. 12. *DACE* and *DACE*-A accuracy by the number of training databases.

In this section, we explore the differences in performance between *DACE* and cost estimation models based on actual cardinality. Specifically, we replace the DBMS-estimated cardinality used by *DACE* with the actual cardinality, namely *DACE*-A. Fig. 12 shows the difference in performance between *DACE* and *DACE*-A on the three test workloads. The experimental setup is the same as the "Training overhead" in Sec. V-D. We can see that *DACE*-A has better performance. On the other hand, *DACE* needs to utilize the general knowledge of 19 databases to perform similarly to *DACE*-A. *DACE*-A uses more accurate general knowledge. However, obtaining the actual cardinality of all predicates in advance in practical applications is impossible. Therefore, our future work will explore how to efficiently improve general knowledge accuracy for *DACE* learning.

VI. Related Work

Cardinality estimators. Cardinality estimation, a core of query optimizers, is also used as a basis for cost estimation in DBMSs. In recent years, ML-based cardinality estimators have received much academic attention [7], [27], [31]. Researchers categorize these methods into two types: (1) query-driven and (2) data-driven. Query-driven cardinality estimator [8], [12], [18], [26], [39] relies on the query or query plan to estimate the number of rows in the result. Its advantage is that it has higher efficiency than data-driven methods, especially for multi-table estimation [27]. However, its drawback is that the model has poor robustness [14], [21]. Therefore, the model's performance is highly dependent on the training workload. Unlike query-driven approachs, data-driven cardinality estimator learn information from data rather than queries or query plans. Therefore, the performance of data-driven methods does not depend on any workload. However, its drawback is low efficiency, especially for multi-table scenarios [27]. In addition, it's challenging to handle scenarios such as data drift.

ML-based DBMS components. In recent years, machine learning has had many applications in DBMSs (AI4DB) [23], [38], [40]. ML-based methods have excelled in many areas, such as index recommendation [3], resource scheduling [16], and query optimization [2], [37]. For many query-driven cardinality estimators [8], [12], [18], [26], [39], they are also able to perform other database optimization tasks simultaneously. TPool [26] can predict both the cardinality and the execution time of a query plan. QueryFormer [39] can perform cardinality and cost estimation, index recommendation, and query optimization. Cost estimation is the basis for many applications. Therefore, proposing an accurate, robust, and efficient cost estimation model is essential. Zero-Shot [8] offers an across-database cost estimation model that transforms the query plan into a directed graph and inference it through message passing. In addition, Zero-Shot utilizes DeepDB's [9] cardinality estimation results to achieve better performance. However, maintaining a cardinality estimation model on each database is problematic. We need to improve the accuracy and efficiency of the across-database cost estimation model on a specific database.

VII. Conclusion and Future Work

In this paper, we build a lightweight and Database-Agnostic cost estimator called *DACE*, which has superior robustness, accuracy, and efficiency compared to existing cost estimation models. Regarding accuracy, *DACE* can outperform WDMs on a particular database even without utilizing any knowledge of that database. Regarding robustness, *DACE* can adapt to various offset scenarios (including data offset, cross-database scenarios, etc.). In addition, we fine-tune *DACE* based on LoRA so that *DACE* can quickly adapt to across-more scenarios. *DACE*'s training and inference efficiency is much higher than existing cost estimation models, making the application of *DACE* in DBMSs more realistic. Finally, *DACE* can act as a pre-trained encoder to help WDMs improve accuracy and robustness and solve the notorious cold start problem.

In future work, a superior and generalized cost estimation model can be essential in DBMSs. Further, we will explore AI-driven DBMSs.

Acknowledgment

This work is partially supported by NSFC (No. 61972069, 61836007 and 61832017), Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021), and Municipal Government of Quzhou under Grant (No. 2022D037, 2023D044), and Key Laboratory of Data Intelligence and Cognitive Computing, Longhua District, Shenzhen.

REFERENCES

[1] X. Chen, H. Chen, Z. Liang, S. Liu, J. Wang, K. Zeng, H. Su, and K. Zheng, "Leon: A new framework for ml-aided query optimization," *Proc. VLDB Endow.*, vol. 16, pp. 2261–2273, 2023.

[2] X. Chen, Z. Wang, S. Liu, Y. Li, K. Zeng, B. Ding, J. Zhou, H. Su, and K. Zheng, "Base: Bridging the gap between cost and latency for query optimization," *Proc. VLDB Endow.*, vol. 16, pp. 1958–1966, 2023.

[3] B. Ding, S. Das, R. Marcus, W. Wu, S. Chaudhuri, and V. R. Narasayya, "Ai meets ai: Leveraging query executions to improve index recommendations," *Proceedings of the 2019 International Conference on Management of Data*, 2019.

[4] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. R. Narasayya, and S. Chaudhuri, "Selectivity estimation for range predicates using lightweight models," *Proc. VLDB Endow.*, vol. 12, pp. 1044–1057, 2019.

[5] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," *ArXiv*, vol. abs/2002.08155, 2020.

[6] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J. rong Wen, J. Yuan, W. X. Zhao, and J. Zhu, "Pre-trained models: Past, present and future," *ArXiv*, vol. abs/2106.07139, 2021.

[7] Y. Han, Z. Wu, P. Wu, R. Zhu, J. Yang, L. W. Tan, K. Zeng, G. Cong, Y. Qin, A. Pfadler, Z. Qian, J. Zhou, J. Li, and B. Cui, "Cardinality estimation in dbms: A comprehensive benchmark evaluation," *Proc. VLDB Endow.*, vol. 15, pp. 752–765, 2021.

[8] B. Hilprecht and C. Binnig, "Zero-shot cost models for out-of-the-box learned cost prediction," *Proc. VLDB Endow.*, vol. 15, pp. 2361–2374, 2022.

[9] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, "Deepdb," *Proceedings of the VLDB Endowment*, vol. 13, pp. 992 – 1005, 2019.

[10] J. E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *ArXiv*, vol. abs/2106.09685, 2021.

[11] Z. Kaoudi, J.-A. Quiané-Ruiz, B. Contreras-Rojas, R. Pardo-Meza, A. Troudi, and S. Chawla, "Ml-based cross-platform query optimization," *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1489–1500, 2020.

[12] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," *ArXiv*, vol. abs/1809.00677, 2018.

[13] V. Leis, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann, "How good are query optimizers, really?" *Proc. VLDB Endow.*, vol. 9, pp. 204–215, 2015.

[14] B. Li, Y. Lu, and S. Kandula, "Warper: Efficiently adapting learned cardinality estimators to data and workload drifts," *Proceedings of the 2022 International Conference on Management of Data*, 2022.

[15] J. Liu, W. Dong, D. Li, and Q. Zhou, "Fauce: Fast and accurate deep ensembles with uncertainty for cardinality estimation," *Proc. VLDB Endow.*, vol. 14, pp. 1950–1963, 2021.

[16] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," *Proceedings of the ACM Special Interest Group on Data Communication*, 2018.

[17] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, "Bao: Making learned query optimization practical," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1275–1288.

[18] R. Marcus and O. Papaemmanouil, "Plan-structured deep neural network models for query performance prediction," *Proc. VLDB Endow.*, vol. 12, pp. 1733–1746, 2019.

[19] B. McCann, J. Bradbury, C. Xiong, and R. Socher, "Learned in translation: Contextualized word vectors," in *Neural Information Processing Systems*, 2017.

[20] G. Moerkotte, T. Neumann, and G. Steidl, "Preventing bad plans by bounding the impact of cardinality estimation errors," *Proc. VLDB Endow.*, vol. 2, pp. 982–993, 2009.

[21] P. Negi, Z. Wu, A. Kipf, N. Tatbul, R. Marcus, S. Madden, T. Kraska, and M. Alizadeh, "Robust query driven cardinality estimation under changing workloads," *Proc. VLDB Endow.*, vol. 16, pp. 1520–1533, 2023.

[22] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[23] A. Pfadler, R. Zhu, W. Chen, B. Huang, T. Zeng, B. Ding, and J. Zhou, "Baihe: Sysml framework for ai-driven databases," *ArXiv*, vol. abs/2112.14460, 2021.

[24] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, pp. 1872 – 1897, 2020.

[25] G. Saxena, M. Rahman, N. Chainani, C. Lin, G. C. Caragea, F. Chowdhury, R. Marcus, T. Kraska, I. Pandis, and B. Narayanaswamy, "Auto-wlm: Machine learning enhanced workload management in amazon redshift," *Companion of the 2023 International Conference on Management of Data*, 2023.

[26] J. Sun and G. Li, "An end-to-end learning-based cost estimator," *Proc. VLDB Endow.*, vol. 13, pp. 307–319, 2019.

[27] J. Sun, J. Zhang, Z. Sun, G. Li, and N. Tang, "Learned cardinality estimation: A design space exploration and a comparative evaluation," *Proc. VLDB Endow.*, vol. 15, pp. 85–97, 2021.

[28] R. Taft, W. Lang, J. Duggan, A. J. Elmore, M. Stonebraker, and D. DeWitt, "Step: Scalable tenant placement for managing database-as-a-service deployments," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 388–400.

[29] C. Tsapelas and G. Koutrika, "Qpseeker: An efficient neural planner combining both data and queries through variational inference."

[30] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Neural Information Processing Systems*, 2017.

[31] X. Wang, C. Qu, W. Wu, J. Wang, and Q. Zhou, "Are we ready for learned cardinality estimation?" *Proc. VLDB Endow.*, vol. 14, pp. 1640–1654, 2020.

[32] Wikipedia. (2022) Partially ordered set. [Online]. Available: https://en.wikipedia.org/wiki/Partially_ordered_set

[33] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacıgümüş, and J. F. Naughton, "Predicting query execution time: Are optimizer cost models really unusable?" *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 1081–1092, 2013.

[34] Z. Wu, P. Yang, P. Yu, R. Zhu, Y. Han, Y. Li, D. Lian, K. Zeng, and J. Zhou, "A unified transferable model for ml-enhanced dbms," in *Conference on Innovative Data Systems Research*, 2022.

[35] Z. Yang, A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica, "Neurocard," *Proceedings of the VLDB Endowment*, vol. 14, pp. 61 – 73, 2020.

[36] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, P. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica, "Deep unsupervised cardinality estimation," *Proc. VLDB Endow.*, vol. 13, pp. 279–292, 2019.

[37] X. Yu, G. Li, C. Chai, and N. Tang, "Reinforcement learning with tree-lstm for join order selection," *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1297–1308, 2020.

[38] J. Zhang, C. Zhang, G. Li, and C. Chai, "Autoce: An accurate and efficient model advisor for learned cardinality estimation," *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pp. 2621–2633, 2023.

[39] Y. Zhao, "Queryformer: A tree transformer model for query plan representation," *Proc. VLDB Endow.*, vol. 15, pp. 1658–1670, 2022.

[40] R. Zhu, Z. Wu, C. Chai, A. Pfadler, B. Ding, G. Li, and J. Zhou, "Learned query optimizer: At the forefront of ai-driven databases," in *International Conference on Extending Database Technology*, 2022.

[41] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, pp. 43–76, 2019.