Efficient Cardinality and Cost Estimation with Bidirectional Compressor-based Ensemble Learning

Zibo Liang[1], Xu Chen[1], Yan Zhao[2], Jiandong Xie[3], Kai Zeng[3], Kai Zheng[1,*,†]
[1]University of Electronic Science and Technology of China, China [2]Aalborg University, Denmark
[3]Huawei Technologies Co., Ltd., China
{zbliang, xuchen}@std.uestc.edu.cn, yanz@cs.aau.dk,
{xiejiandong, kai.zeng}@huawei.com, zhengkai@uestc.edu.cn

***Abstract*—Query optimization is of great importance for the performance of a database, in which cardinality and cost estimation have a pivotal role. To enable accurate cardinality and cost estimation, we propose a novel framework based on bidirectional compressor and ensemble networks called BICE. In particular, we design a feature extractor composed of four sub-encoders, which can extract various types of information in a query plan tree and hybrid learning strategies for encoding. We encode joins based on a graph embedding method and design parallel networks for filters to improve the encoding efficiency. Then we propose a bidirectional LSTM-based compressor to learn the encoding and obtain fixed-length vectors, reducing the learning difficulty of the estimation model. Finally, we propose different data sampling strategies based on Bayesian neural networks and active learning, and an ensemble model is established based on transfer learning, which enables accurate estimation and adaptation to large-scale data queries. Extensive experiments offer insight into the effectiveness and efficiency of the proposed framework.**

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

I. Introduction

Cardinality and cost estimation are crucial in query optimization, a feature of many database management systems and determined by a query optimizer [1]. Incorrect estimates may lead to sub-optimal plans or even produce poor plans that have catastrophic effects on database performance [2]. Many traditional estimation methods, e.g., histogram-based cardinality estimators, are based generally on the assumptions that the data follow a certain statistical distribution and all attributes are independent of each other [3]. However, in practice, a dataset is often highly random [2], [4], [5] without following any distributions, which limits the feasibility of traditional cardinality and cost estimation methods.

Recent years have witnessed the development of learning models as advanced cardinality and cost estimation tools [6]–[14]. Although data-driven cardinality estimation techniques [6], [15], [16] yield more precise estimates, query-driven methods [7], [8], [10] offer advantages in terms of computational efficiency and storage space requirements. Besides, their seamless integration into query optimizers also fosters extensive application across various contexts. In particular, the query-driven method can predict the execution cost of a query plan as the basis for query optimization. So it is difficult for the data-driven to replace the query-driven with the existing advantages completely [17]–[19]. In this work, we will go further in this direction and focus on query-driven models.

Existing query-driven models can be classified into two categories: *query-statement-based models* [8], [11], [12] and *query-plan-tree-based models* [7], [10], [20]. The former takes a query statement as input and calculates the cardinality of the query result without relying on any other information. Taking the same input, i.e., a query statement, the latter uses the query plan tree [21] generated by the optimizer of a DBMS to estimate the query cardinality. It is worth noting that this method does not require the execution of query statements in the database and is efficient, as DBMS uses traditional estimation methods to obtain plans [22]. Although these plans may be suboptimal, we can use the information embedded in them for cardinality estimation. Therefore, the learning models based on the query plan tree can utilize more helpful information and generate a more accurate cardinality estimation with little extra computational cost. Although query-based learning models have many advantages, some limitations prevent them from becoming silver bullets for query optimization.

Limitation I: paucity of extracted information. Extracting information from query statements is a crucial step in query-driven methods, which directly affects the accuracy of the estimation. For example, the multi-set convolutional network (MSCN) [8] directly adopts the one-hot encoding for joins, leading to a serious sparsity problem and increasing the model's learning difficulty. TPool [7], an end-to-end learning-based cost estimator, extracts the information in the query plan tree, but the one-hot encoding method it adopts is still simple, which cannot make good use of the rich information contained in the query plan tree.

Limitation II: high computational cost. The training and the testing efficiencies of the cardinality and cost estimation model are two main factors limiting the feasibility of query-driven models [4], [5], [23]. For example, MSCN [8] uses the idea

*Corresponding author: Kai Zheng.
†Kai Zheng is with Yangtze Delta Region Institute (Quzhou), School of Computer Science and Engineering, and Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China.

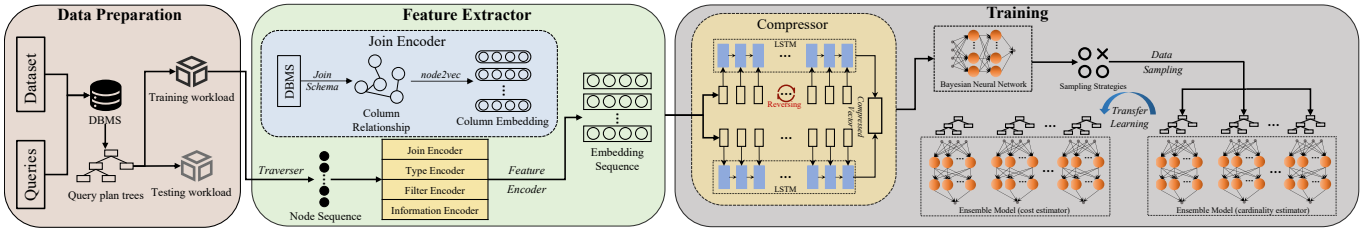


Fig. 1: BICE Framework Overview

of Deep Set to compute tables, joins, and filters. QPPNet [10] designs a neural network based on a tree structure, where the upper layer's network takes the lower layer's output as input. The encoding method of TPool generates a lot of redundant information, and the representation layer also contains many parameters, resulting in high computational complexity. The above models require high training or inference costs, which hinders them from applying to more real-world scenarios.

Limitation III: poor data adaptability. Although neural networks have been used successfully in many fields [24], they apply simply in cardinality and estimation. For instance, MSCN [8] only uses a four-layer linear neural network as the output model, and QPPNet [10] equip with a simple five-layer linear neural network for each node. However, a single model cannot fit all kinds of queries.

To tackle these limitations, we propose a cardinality and cost estimation framework based on Bidirectional Compressor-based Ensemble Learning (BICE). Specifically, we design four sub-encoders to encode various information in the query plan tree efficiently. Then we use a compressor to reduce the learning difficulty of the estimation model and build an ensemble model based on Bayesian neural networks and active learning to enhance the data adaptation of the estimation model. Moreover, we employ transfer learning to efficiently obtain the cost estimation model to estimate both the cardinality and the cost.

Our contributions can be summarized as follows:

- We propose a novel framework, namely Bidirectional Compressor-based Ensemble Learning (BICE), which can estimate cardinality and cost simultaneously and efficiently.
- We propose a query plan tree-based feature extractor consisting of four sub-encoders and use graph embedding and parallel networks to encode joins and filters efficiently. Then we design a compressor based on a bidirectional network that can learn the feature extractor's encoding output and reduce the estimation model's learning difficulty, which solves *Limitations I and II*.
- We establish an ensemble model based on Bayesian neural networks and active learning, where estimating cardinality and cost estimations combine through transfer learning. This model can estimate the cardinality and cost accurately and improve the efficiency, which solves *Limitation III*.
- We report on experiments using the public JOB and TPC-H datasets, offering evidence of the effectiveness and efficiency of the proposed framework.

The remainder of the paper organizes as follows. Section II covers preliminaries. The BICE framework is detailed in Section III, followed by coverage of experimental results in Section IV. Section V surveys related work, and Section VI concludes the paper.

II. Problem Statement

Definition 1 (Query Statement): Q represents a set of query statements, and q is a query statement in Q. q consists of several elements in a join set $J = \{J_1, J_2, ..., J_m\}$ and a filter set $F = \{F_1, F_2, ..., F_n\}$ (e.g., SELECT COUNT(*) FROM $Relation_1, Relation_2$ WHERE J_1 AND J_2 AND F_1 AND F_2;), m and n denote the numbers of joins and filters, respectively. $J_i (i \in [1, m])$ is a join condition for two columns (e.g., $t.id = mc.movie\_id$) and $F_i (i \in [1, n])$ is a filter condition for a single column (e.g., $t.id > 4$).

Definition 2 (Query Plan Tree): The query plan tree is an execution plan estimated by the DBMS using traditional algorithms. It consists of several nodes, each with a unique type (e.g., $HashJoin$, $NestedLoopJoin$, etc.), and may contain the join or filter conditions.

Problem Statement. Given an unseen query statement q, our problem estimates its cardinality $C(q)$ and cost $T(q)$ by trained f, where $f(q) = (C(q), T(q))$, $C_r(q)$ denotes the real cardinality, and $T_r(q)$ denotes the real execution time, and make $C(q)$ and $T(q)$ close to the real cardinality $C_r(q)$ and the real execution time $T_r(q)$.

III. Framework and Methodology

We propose a framework, namely BICE, to estimate cardinality and cost. We first overview the framework and then provide specifics on each component.

A. Framework Overview

Fig. 1 shows the framework of BICE, which consists of five components, i.e., data preparation, feature extractor, compressor, and training.

Data Preparation. The data preparation component sets up a DBMS environment and imports a dataset. This component generates the query statements based on the dataset and obtains the corresponding query plan trees (e.g., EXPLAIN command in PostgreSQL) through the DBMS.

Feature Extractor. We design a join encoder in the feature extractor, which can embed all columns based on their relationships. Then a feature encoder is given, including four sub-encoders, i.e., a trained-to-join encoder, a type encoder, a filter encoder, and an information encoder. The feature encoder encodes a sequence of nodes obtained by a traverser into a node embedding sequence.

Compressor. The node embedding sequence feeds into the compressor component composed of two LSTMs, where one LSTM enhances its learning ability for different historical information (of the nodes) by inverting the sequence. As a result, we generate a series of compressed node embeddings.

Training. In the training phase, we obtain the trained Bayesian neural networks and then use different active learning strategies to sample the data. For different sampling results, we use transfer learning to get multiple models with additional capabilities, based on which ensemble learning is employed to learn the cardinality and cost.

B. Data Preparation

BICE needs to obtain the prior knowledge from DBMS, i.e., using the underlying query optimizer of DBMS to produce query execution plans for query statements [21]. Getting the query plan trees according to the above method has the following advantages:

High Computational Efficiency. The query optimizer uses a coarse-grained sampling method (e.g., histogram) to estimate statistics to generate a query plan tree, which achieves high computational efficiency.

Prior Knowledge Acquisition of DBMS. The query optimizer has been studied for decades [25], which contains many basic rules for statistics computation [26]. For example, PostgreSQL calculates the cost according to the weighted sum of CPU cycles and IO usages [25]. Although they are not closely related to the actual cardinality and cost, BICE can leverage those estimations made by the query optimizer to make accurate predictions. We will analyze this advantage further in Section III-C.

C. Feature Extractor

The proposed feature extractor aims to encode the query plan tree obtained in Section III-B into a corresponding vector sequence. Specifically, we get the node sequence of the query plan tree by a *traverser*, and then each node is input into the feature encoder that consists of four sub encoders, as shown in Fig. 1. Finally, we splice the obtained vector of each node to get a vector sequence.

1) Traverser: The first job that *feature extractor* does after accepting the query plan tree is to obtain its corresponding node sequence through traversing. We choose Depth First Search (DFS) [27] as the traversal method for *BICE*. The sequence we obtain with different traversal methods will impact the final result because some tree nodes may have complex relationships with their child nodes and parent nodes [21]. Using a universal traversal method to represent a plan tree by a vector takes a lot of work. Instead, we use a bidirectional method to solve that in Section III-D.

2) Join encoder: Section II decomposes a query statement into several joins J with filters F. The role of *join encoder* is to encode join keys (e.g., $t.id = mc.movie\_id$). TPool [7], QPPNet [10] and MSCN [8] use one-hot encoding for join types and columns, but they fail to represent columns with underlying relationship (e.g., the foreign key constraint). Furthermore, the above encoding methods lead to sparse problems in complex joins, increasing the learning difficulty.

The success of graph embedding algorithms in many fields [28] inspires us that if all relationships between columns in the database are extracted and considered as undirected graphs for embedding, it not only solves the sparsity problem but also makes the encoding contain more rich information.

Based on the above observations, we can use an undirected graph to represent the relation model for a database. Specifically, we leverage a node2vec algorithm [29], [30] to embed all columns, defined as follows: $e_i = node2vec(c_i)$, where c_i and e_i denote the ith column and its corresponding embedding in the database, i ($1 \leq i \leq n_c$) is an integer, and n_c is the number of columns.

After obtaining the embedding corresponding to each column by $node2vec$, as shown in Fig. 2, next, we use column embeddings to represent their join keys.

We use the prior knowledge of DBMS to decompose several joins in a complex query statement into several nodes [21], [25], [31], where the join condition of each node is only composed of two columns, so we can directly connect the embeddings of the columns to obtain fixed-length $E_J(J)$ corresponding to the joins J. It is worth noting that the order of the columns in the join condition does not affect $E_J(J)$, e.g., $t.id = mc.movie\_id$ and $mc.movie\_id = t.id$, where $t.id$ and $mc.movie\_id$ are the same join condition in the *join encoder*. When some nodes do not contain join conditions, we use an all-zero vector to represent this situation.

3) Type Encoder: The *type encoder* encodes the types of nodes (e.g., $HashJoin$, $NestedLoopJoin$, etc.), and the node types in query plan trees have the following properties:

Limited Number of Types. We consider nine node types, including 'Sequential Scan', 'Index Scan', 'Bitmap Index Scan ', 'Bitmap Heap Scan', 'Index Only Scan', 'Hash Join', 'Merge Join', 'Nested Loop Join' and 'Hash'. In addition, we can easily add other node types to *BICE*.

No Obvious Relationship Between Different Types. The operators in DBMSs are independent of each other. Therefore, it is feasible to presuppose that all node types do not have any relationship with each other and then to inscribe their possible relationships by models.

Based on the above observations, we use one-hot to represent the node types, i.e., obtaining $E_T(T)$ corresponding to node types T. As shown in Fig. 2, a node type (e.g., Seq Scan) is input into the *type encoder* to obtain its corresponding encoding, which has nine bits since we consider a total of nine node types. Note that we can easily extend the type encoder with more classes based on different execution engines.

4) Filter Encoder: Filters (e.g., $t.kind\_id < 4$) are one of the most complex conditions in nodes, which encode three data types: columns, operators (i.e., $>$,$<$, and $=$), and values. Previous one-hot encoding studies [7], [10] are not expressive enough to represent such complex information, hindering learning progress. Fauce [12] uses ranges to represent the information in filters and achieve more accurate experimental

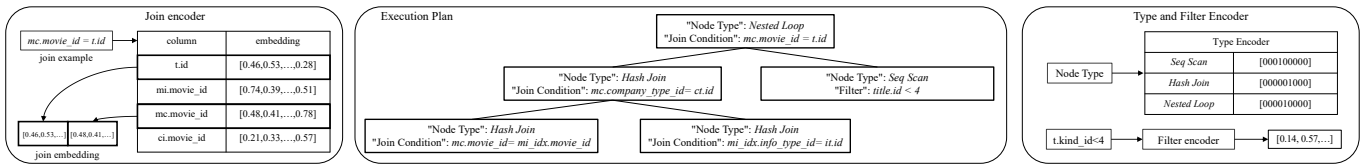


Fig. 2: An Example of Encoding

results in the application scenario based on query statements. However, directly using range encodings for all nodes in the query plan tree will lead to difficulties in model learning because the model needs to simultaneously learn the range representations of many nodes in a query plan tree. Based on the above considerations, we design a *filter encoder* based on range representations and parallel networks, which can represent the information in the filter well and reduce the learning difficulty of the model.

DNNs exhibit success across various fields but need help with multi-categorical information. Parallel DNNs tackle this issue by independently processing distinct information categories in separate networks. This approach excels in multi-categorical data scenarios [32]. With its maximum and minimum column values, range representation exemplifies such classification. The procedure unfolds as follows:

First, we extract the range representations $[V_1^{max}, V_2^{max}, ..., V_{n_c}^{max}]$ and $[V_1^{min}, V_2^{min}, ..., V_{n_c}^{min}]$ of all columns from the dataset, where V_i^{max} and V_i^{min} are the maximum and minimum values of the ith column, $i \in [1, n_c]$, and n_c is the number of columns. We use two vectors V^{max} and V^{min} to store maximum and minimum values, respectively. For three different operators in the filter, we design the corresponding encoding rules as follows:

a) $c_i > value$*:* Let V_i^{min} in the V^{min} vector be $value$.

b) $c_i < value$*:* Let V_i^{max} in the V^{max} vector be $value$.

c) $c_i = value$*:* Let V_i^{min} in the V^{min} vector be $value$, and V_i^{max} in the V^{max} vector be $value$.

The encoding of the filter is obtained by reading all the operators in the node and modifying V^{max} and V^{min} according to the above rules. After that, the two parts (i.e., maximum and minimum) of the encoding are input into two parallel networks to get the final encoding of the filter:

$$R^{max}(F) = ReLU(\acute{R}^{max}(F) * W_{max} + b_{max}) \quad (1)$$

$$R^{min}(F) = ReLU(\acute{R}^{min}(F) * W_{min} + b_{min}) \quad (2)$$

where $\acute{R}^{max}(F)$ and $\acute{R}^{min}(F)$ are the vectors obtained by modifying V^{max} and V^{min}, respectively. W_{max} and W_{min} denote the weights of neural networks for the maximum and minimum values, respectively, and b_{max} and b_{min} denote the bias of neural networks for the maximum and minimum values, respectively. $R^{max}(F)$ and $R^{min}(F)$ are the encodings of maximum and minimum values in the range representation, respectively. The method to get the final encoding is defined as follows:

$$\acute{E}_F(F) = [R^{max}(F), R^{min}(F)] \quad (3)$$

$$E_F(F) = ReLU((\acute{E}_F(F) * W_F + b_F)) \quad (4)$$

where W_F and b_F denote the weights and bias of the neural network for the encoding, respectively, and $E_F(F)$ is the final encoding corresponding to the filter F. Same as the *join encoder* in Section III-C2, we use a full range vector to represent the case where no filter exists in the node.

5) Information Encoder: We proceed to detail the *information encoder*.

The query plan tree contains the prior knowledge generated by the DBMS using traditional estimation methods [21]. We fully utilize it to make the model stand on the shoulders of the DBMS for estimation. Therefore, we designed an *information encoder* to encode the prior knowledge generated by DBMS (e.g., PostgreSQL) as follows: $E_I(I) = [C'(p), T'(p)]$, where I is the statistic information in the nodes of the query plan tree, $C'(p)$ and $T'(p)$ represent the estimated cardinality and cost of the query plan p (including sub-plans) by the DBMS, respectively, and $E_I(I)$ denotes the encoding corresponding to the statistic information I.

The *feature encoder* splices the encodings output by four subencoders, i.e., $\acute{E}_Q(q) = [E_J(J), E_T(T), E_F(F), E_I(I)]$. To make the encoding more beneficial for the calculation of subsequent models, we design a layer of neural network in the *feature encoder* to learn $\acute{E}_Q(Q)$, which is defined as follows: $E_Q(q) = ReLU(\acute{E}_Q(q) * W_Q + b_Q)$, where W_Q and b_Q represent the weight and bias of the neural network, and $E_Q(q)$ is the encoding corresponding to the query statement q. The *feature extractor* extracts rich information from the query plan tree and represents information into a learnable dense feature vector, which lays a foundation for the subsequent model.

D. Compressor

We acquire q's encoding $E_Q(q)$ using the *feature extractor*. However, directly inputting it into networks (e.g., DNN) for estimation poses issues.

Rich Information. $E_Q(q)$ consists of complex vector sequences, leading neural networks to local optima or overfitting [24].

Temporal Relationship. The node sequence, discussed in Section III-C1, contains temporal information, which is challenging for traditional neural networks to learn [33].

TPool [7] uses LSTM for node sequences but faces forgetfulness and learning issues [34]. QPPNet [10] employs a tree-structured convolutional neural network, accurately representing execution order but losing information with added layers [21], [31] (Section III-C4).

Taking advantage of bidirectional RNNs [35], [36], we implement a two-layer bidirectional LSTM model to capture

the encoding sequence within our approach effectively. The reversed LSTM is represented as $h = \overleftarrow{LSTM}(s; W_l)$, and the second LSTM is defined as $\widetilde{h} = \widetilde{LSTM}(\widetilde{s}; W_{\widetilde{l}})$. The LSTMs operate independently, enabling parallel computation and minimal time overhead. The final compressed encoding is $E_C(s) = [h, \widetilde{h}]$, compensating for potential information loss in longer sequences. The compressor outputs a fixed-length compressed vector $E_C(E_Q(q))$, denoted as E, when the context is clear.

E. Ensemble Learning

Section III-D tackles two estimation challenges, but accurate actual cardinality $C_r(q)$ and execution time $T_r(q)$ estimation remains complex. Existing models disregard data distribution characteristics, including diverse data types/ranges and skewed distribution.

Ensemble learning models excel by integrating multiple models' results [37], [38], addressing two primary issues:

Problem 1. Data classification based on characteristics.

Problem 2. Developing models with varying capabilities.

We propose an ensemble learning component using a Bayesian model to train the *compressor*, applying diverse, active learning strategies and designing an ensemble model for cardinality and cost estimation.

1) Bayesian Model: The data often possess strong randomness, and traditional neural networks presuppose that the data will obey a specific mathematical distribution, which usually does not correspond to the actual situation. In recent years, Bayesian neural networks (BNN) have achieved success in many fields [39]. It is a stochastic neural network with a probability distribution over the weights and is to find the posterior distribution of parameters, which can measure uncertainty in the neural network. We use the MC_Dropout [40] method to build a BNN as follows: $f_B(q|W_B) = C(q)$, where W_B represents weights of neural networks. We use a 3-layer fully connected neural network with the MC_Dropout method to build a BNN and use $C(Q)$ to estimate the cardinality of the BNN output. We will introduce the process of estimating cost $T(Q)$ in Section III-E4. f_B accepts the fixed-length vectors output by taking *compressor* as input and outputs the estimated cardinality. The parameters in f_B and *compressor* can be updated by calculating the loss and backpropagation. We use the $q\text{-}error$ metric [8] for training and testing. Based on the $q\text{-}error$ function, we define the loss function as follows:

$$Loss = q\text{-}error(C_r(q), \frac{1}{n_B}\sum_{i=1}^{n_B} f_B(q|W_B)) \tag{5}$$

where n_B represents the number of computations of the Bayesian neural networks f_B for the same sample. Since solving the posterior distribution is a time-consuming task [39], we use $\frac{1}{n_B}\sum_{i=1}^{n_B} f_B(q|W_B)$ to approximate the mathematical expectation of $f_B(q|W_B)$, and then update the parameters in f_B and *compressor* through backpropagation based on the $Loss$ function.

Finally, we finish training f_B and *compressor*, which enables the *compressor* to adapt to more data types by introducing uncertainty into neural networks and lays the foundation for data sampling in Section III-E2.

2) Data Sampling: This section proposes a solution to *Problem 1*. Active learning aims to explore how to learn models with less training data, i.e., finding which data is more meaningful for training and obtaining data samples with specific characteristics by designing different strategies. We design a total of four strategies for data sampling as follows:

Strategy 1: Uncertainty of BNN. We can obtain the uncertainty of the data based on the characteristics of BNN, i.e., finding data samples that generate an output with considerable fluctuation. We use variance to describe this uncertainty, defined as follows:

$$Var(q) = \frac{1}{n_B}\sum_{i=1}^{n_B}(f_B(q|W_B) - \overline{f_B(q|W_B)})^2 \tag{6}$$

where $\overline{f_B(q|W_B)}$ is $\frac{1}{n_B}\sum_{i=1}^{n_B} f_B(q|W_B)$. After that, we select the top $k\%$ of the data with greater variance to obtain a data sampling set.

Strategy 2: Upper Confidence Bounds with Uncertainty. We obtain data samples with uncertainty by *Strategy 1* but do not take into account the bias (i.e., $q\text{-}error$) in the estimation. The Upper Confidence Bounds (UCB) is one of the classical algorithms in reinforcement learning. In our problem, the reward value is the opposite or multiplicative inverse of $q\text{-}error$ (i.e., minimizing $q\text{-}error$), defined as follows:

$$Ucb(q) = Var(q) + Max(q_1, q_1, ..., q_{n_B}) \tag{7}$$

where $q_i (i \in [1, n_B])$ is the $q\text{-}error$ corresponding to $f_B(Q|W_B)$. We select the data with larger $q\text{-}error$ for retraining to maximize the reward value. Since some query retraining brings only a small improvement, we retain uncertainty to comprehensively judge which data is more valuable for training [41]. As in *Strategy 1*, we select the top $k\%$ of the data with the larger value in $Ucb(Q)$.

Strategy 3: Diversity Sampling Combined with Strategy 1. Diversity sampling aims to obtain a more affluent sample of data, which is a more exploratory strategy than *Strategy 1* and *Strategy 2*, and can effectively avoid the problem of a single sample type based on indicators alone. Therefore, we design *Strategy 3* by combining the existing diversity sampling methods [42], [43], i.e., first clustering the fixed-length vectors outputted by *compressor* using a k-means algorithm, and then selecting the top $k\%$ of data in each category with greater uncertainty (obtained by Var function).

Strategy 4: Diversity Sampling Combined with Strategy 2. Similar to *Strategy 3*, in *Strategy 4*, the top $k\%$ of data with larger value (obtained by Ucb function) in each cluster obtained by k-means are selected.

We denote the set of data samples obtained through *Strategies 1 to 4* as $D = \{D_1, D_2, D_3, D_4\}$. Besides, we use min-max normalization to eliminate the effect of different data ranges of variance and $q\text{-}error$ on $Ucb(q)$.

3) Ensemble Model: Problem 2 can be solved well by ensemble learning. We use the Bagging method in ensemble learning [38] to obtain different models by training on diverse data sample sets D, and fuse these models for different data

types to form an ensemble model for cardinality estimation, defined as follows: $C(q) = \frac{1}{m}\sum_{i=1}^{m} f_{C_i}(q|W_{C_i})$, where m is the number of models in ensemble learning. We have extracted four data sample sets, i.e., $m = 4$, and W_{C_i} represents the weight of the i-th cardinality estimation model.

4) Transfer Learning: Since the cardinality and cost often strongly connect with each other [26], [44] and the ensemble cardinality estimation model is established, we can utilize transfer learning to build a cost estimation model. Therefore, we adopt a full-parameter update strategy of inductive learning in transfer learning [45], [46] to obtain the cost estimation model. $f_{C_i}(1 \leq i \leq m)$ has the ability to estimate the cardinality. We use the relationship between the cardinality and cost to update the parameters of f_{C_i} using the transfer learning strategy based on the data samples obtained in Section III-E2, i.e., $f_{C_i} \longrightarrow f_{T_i}$.

Finally we can define f as follows:

$$f = (\frac{1}{m}\sum_{i=1}^{m} f_{C_i}(q|W_{C_i}), \frac{1}{m}\sum_{i=1}^{m} f_{T_i}(q|W_{T_i})) \tag{8}$$

where W_{T_i} represents the weight of the ith estimation model.

IV. Experimental Evaluation

A. Experimental Setup

1) Datasets: We select four commonly-used query statement sets based on IMDB [2] and TPC-H [47] datasets. IMDB includes 22 tables with a total of 3.6GB of data. We set the scale of TPC-H to 10, i.e., including eight tables with a total of 10GB of data. The query workloads are as follows: a) JOB-light: JOB-light [8] based on the IMDB dataset has 70 queries, including 1-4 joins. b) Scale: Scale [8] based on the IMDB dataset, which has 500 queries, including 0-4 joins. c) Synthesis: Synthesis [8] based on the IMDB dataset, which has a total of 5000 queries, including 0-2 joins. d) TPC-H (test): TPC-H (test) has 2200 queries generated by qgen [47] and based on 22 templates provided by TPC-H, including 1-8 joins.

The above four datasets are our workloads for testing. For IMDB, the training data is 100,000 queries provided by MSCN [8], which includes 0-2 joins. For TPC-H, we generate 22,000 queries for training based on the same templates.

2) Evaluation Methods: We compare *BICE* with various representative cardinality and cost estimators.

PostgreSQL [31]: PostgreSQL uses histogram and heuristic search to estimate the cardinality and cost. Since the unit of cost estimated by PostgreSQL has no practical significance, we only use it for cardinality estimation. **MSCN** [8]: MSCN is a cardinality estimation model based on query statements. It uses the idea of Deep Set to process tables, joins, and filters. **QPPNet** [10]: QPPNet is a cost estimation model based on the query plan tree, which designs different neural networks for computing the information of different types of nodes. **TPool** [10]: TPool designs a feature extraction algorithm based on the binary tree, encodes information such as filters, joins, and node types, and adds Sample bitmap encoding to improve the accuracy of the estimation. **QueryFormer** [20]: QueryFormer uses a tree-structured model combined with the attention mechanism, which applies to multiple estimation problems in database optimization.

3) Metrics: We use $q\text{-}error$ [8] to evaluate the accuracy of cardinality and cost estimation. We also evaluate the efficiency of the models, including the training and testing time.

4) Implementaion Details: The implementation details of our experiments are as follows:

Parameters Setting. The length of the embedding vector obtained by $node2vec$ is 32, i.e., $|e_i| = 32$. The number of neurons for W_{min}, W_{max}, and W_P in the *filter encoder* is 32, 32, 64, respectively. We process the range representation with min-max normalization before input into the neural network. In the *compressor*, the numbers of neurons in the hidden layers of $LSTM$ and $\widetilde{LSTM}$ are 139 (determined by the output of the feature extractor), and set n_s to 20 (which we will cover in Batch Training), so the length of the vector output by the *compressor* is 278. The numbers of neurons in all layers of the Bayesian neural networks (i.e., W_B) and the ensemble model (i.e., W_C and W_T) are [278, 128, 64]. The last layer is processed using a Sigmoid nonlinear activation function. The proportion k of data sampling by active learning in Section III-E2 is 25 (i.e., we sample 25% of the data each time). Finally, we use 10% of the training data as the validation data, update the parameters by the Adam algorithm [48], and set the learning rate to 0.0001.

Batch training. Training efficiency is one of the essential metrics to measure the model QPPNet [10] and TPool [7] can only encode binary trees. In contrast, QPPNet has to use different networks for computation when facing different shapes of trees. Our BICE can accept arbitrarily shaped trees as input (based on the nature of the traverser in Section III-C1), but the lengths of sequences obtained by the traverser are not fixed. Thus we propose two batch training methods.

The first training method is sequence-based batch training. Considering that QPPNet classifies trees of different shapes and performs batch training in each category, we classify sequences according to their lengths, perform batch training on sequences of the same length, and use fewer data categories compared to QPPNet (because trees of different shapes may have the same numbers of nodes).

The second training method is padding-based batch training. This method is widely used in many fields to obtain fixed-length sequences by padding unequal sequences. We use this method for batch training, where the batch size is 128.

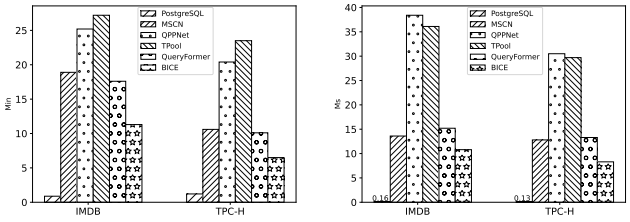
Hardware and Platform. The hardware devices and platforms used in our experiments are as follows: the CPU is Intel(R) Xeon(R) CPU E5-2650 v4@2.20GHz, the GPU is GeForce GTX 1080 Ti, the DBMS selected is PostgreSQL 10.17, the algorithms are implemented based on Python 3.7.10, and the deep learning framework is Pytorch 1.8.1.

B. Experimental Results

1) Overall Accuracy: Tab. I shows the $q\text{-}error$ of BICE and other models in terms of cardinality and cost estimation,

TABLE I: Cardinality and Cost Errors

Cardinality Errors																								
Datasets	JOB-light						Synthetic						Scale						TPC-H (test)					
	50th	*90th*	*95th*	*99th*	*max*	*mean*	*50th*	*90th*	*95th*	*99th*	*max*	*mean*	*50th*	*90th*	*95th*	*99th*	*max*	*mean*	*50th*	*90th*	*95th*	*99th*	*max*	*mean*
PostgreSQL	7.93	164	1104	2912	3477	174	1.69	9.57	23.9	465	373901	154	2.59	200	540	1816	233863	568	2.35	42.2	257	478	1542	20.3
MSCN	3.82	78.4	362	927	1110	57.9	1.18	3.32	6.84	30.5	1332	2.89	1.42	37.4	140	793	3666	35.1	2.52	14.4	26.1	57.2	102	18.7
QPPNet	3.52	44.6	134.5	205.3	247	28.4	1.25	4.23	7.59	28.4	294	2.48	1.47	42.5	137	453	1243	26.4	2.13	7.46	18.5	30.4	87.3	8.46
TPool	3.73	50.8	157	256	289	24.9	1.20	3.21	6.12	25.2	357	2.87	1.43	**38.8**	139	469	1892	28.1	2.25	33.2	42.6	84.9	108	13.5
QueryFormer	**2.31**	25.8	42	81	109	23.3	**1.14**	3.31	6.83	29.5	451	2.65	1.42	45.8	153	409	952	25.7	2.27	8.53	**16.2**	35.8	93	8.41
BICE	2.62	**10.78**	**11.29**	**20.89**	**33.56**	**5.10**	1.23	**2.54**	**6.05**	**23.5**	**236**	**2.30**	**1.37**	40.8	**148**	**320**	**549**	**19.4**	**2.22**	**6.84**	17.5	**22.6**	**64.2**	**6.54**
Cost Errors																								
Datasets	JOB-light						Synthetic						Scale						TPC-H (test)					
	50th	*90th*	*95th*	*99th*	*max*	*mean*	*50th*	*90th*	*95th*	*99th*	*max*	*mean*	*50th*	*90th*	*95th*	*99th*	*max*	*mean*	*50th*	*90th*	*95th*	*99th*	*max*	*mean*
MSCN	3.03	54.5	184	397	549	17.3	2.01	17.73	33.4	108.4	823	8.34	1.72	10.8	32.5	245	593	7.31	4.12	14.53	38.47	49.5	169	6.74
QPPNet	1.67	12.5	21.4	83	113	5.21	1.44	4.49	**8.18**	48.1	516	3.77	**1.31**	6.38	14.7	59.4	274	4.57	**1.29**	4.83	8.36	16.4	41.9	3.51
TPool	1.85	13.2	22.9	95	123	5.81	1.49	4.33	10.2	55.8	624	4.16	1.56	**5.56**	12.2	68.6	254	4.41	1.94	9.23	22.1	34.1	94.7	4.57
QueryFormer	**1.54**	16.3	29	126	237	11.5	**1.16**	4.03	10.5	45.8	**294**	**1.5**	1.47	5.93	13.8	53.1	219	4.52	1.50	5.81	9.64	15.8	37.5	4.01
BICE	1.74	**11.5**	**17.3**	**52**	**73**	**3.78**	1.39	**3.58**	12.4	**37.1**	385	3.25	1.58	5.84	**9.57**	**43.7**	**187**	**3.86**	1.54	**4.23**	**6.38**	**13.7**	**32.8**	**3.18**



(a) Average training time (b) Average testing time

Fig. 3: Average Training and Testing Time

from which we draw some conclusions as follows:

The cardinality estimation accuracy of the learning model is significantly higher than PostgreSQL, and BICE outperforms PostgreSQL in terms of *mean* metrics by 33 times, 66 times, 28 times, and two times on the four workloads, respectively. The models based on query plan trees (i.e., QPPNet, TPool, QueryFormer, and BICE) outperform those based on query statements (i.e., MSCN).

In the cardinality estimation, BICE outperforms the optimal model (i.e., QueryFormer) on the four workloads by 3.5 times, 15%, 32%, and 28% for the *mean* metric, and by 2.3 times, 91%, 73%, and 45% for the *max* metric, respectively. In the cost estimation, BICE outperforms QueryFormer on JOB-light, Scale, and TPC-H (test) by two times, 17%, and 26% for the *mean* metric and by 2.2 times, 17%, and 14% for the *max* metric, respectively.

The main reason for the better performance of the query plan tree-based estimation model is its ability to exploit richer information and prior knowledge of the DBMS, as analyzed in Section III-C, and thus perform better on many queries, incredibly complex queries. Almost all models (including PostgreSQL) perform better on TPC-H workloads because TPC-H is based on algorithmically generated datasets [47], while IMDB is actual data collected from life with more vital uncertainty, so the estimation of workloads based on the IMDB dataset is more difficult [2].

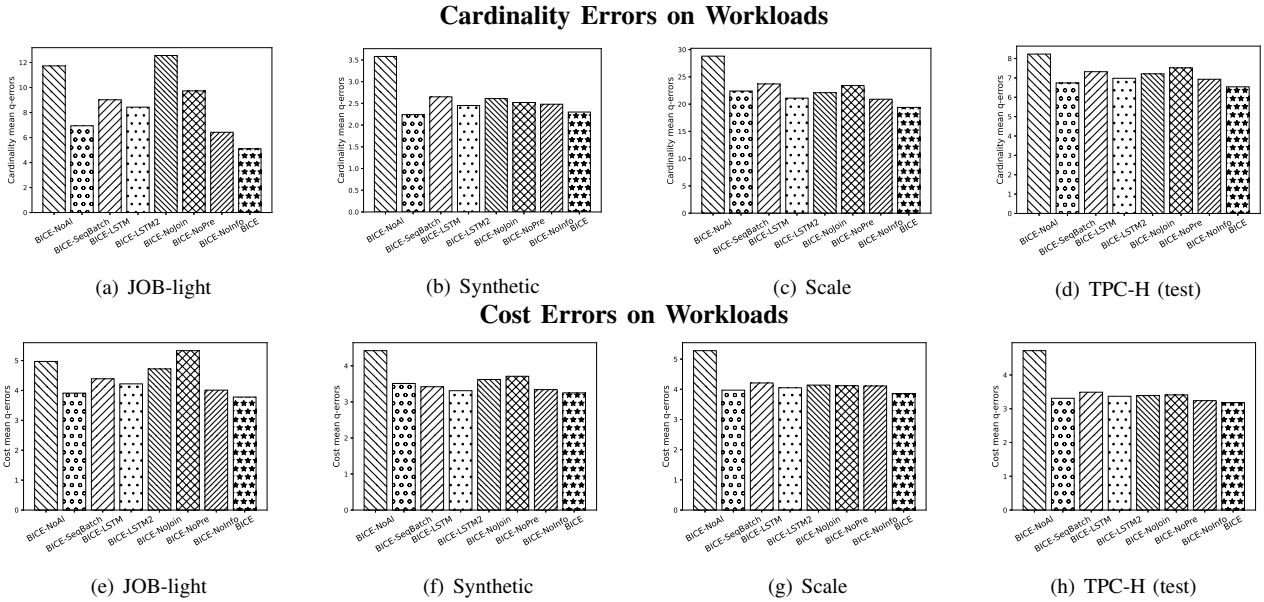
QueryFormer uses linear networks to learn the encoding of the query but does not learn the information contained in the database. QPPNet and TPool are close in estimation accuracy for many workloads. Still, they use a one-hot encoding method for joins and filters, which makes the encoding sparse and increases the learning difficulty of the model for many queries. BICE encodes different types of information through four sub-encoders and applies *compressor* to obtain fixed-length vectors to reduce the difficulty of model learning further, and finally builds an ensemble model that can adapt to a broader range of data types, so BICE is better at estimating many queries including complex ones.

TABLE II: End-to-End Performance

	End-to-End Time	Exec. + Plan Time	Improvement
PostgreSQL	4.93h	4.93h + 5s	0.0%
TrueCard	4.27h	4.27h + 5s	13.4%
MSCN	4.67h	4.67h + 24s	5.3%
QPPNet	4.62h	4.62h + 41s	6.2%
TPool	4.65h	4.65h + 46s	5.6%
QueryFormer	4.57h	4.57h + 28s	7.3%
BICE	**4.51h**	**4.51h + 19s**	**8.4%**

We use the CEB framework to replace cardinality estimates obtained from PostgreSQL and carry out end-to-end evaluations. Tab. II displays the aggregated results. As the data reveals, apart from TrueCard, BICE showcases the most pronounced improvement and the shortest plan time. In contrast, QPPNet and TPool also use query plan trees, which generate 215% and 242% of BICE's time, respectively.

2) Efficiency: Fig. 3 shows BICE's average training time and testing time with other models for all queries. In the testing phase, IMDB includes three workloads, i.e., JOB-light, Synthetic, and Scale, and TPC-H has one workload, i.e., TPC-H (test). We use the ANALYZE command to represent PostgreSQL's learning of IMDB and TPC-H, and use the EXPLAIN command to obtain testing results. We record the training and testing times for the rest of the models. We can see that the model's efficiency based on query statements is significantly higher than those found on query plan trees. MSCN is the most time-consuming model based on query statements because its Deep Set application and bitmap calculation consume extra time. The upper layer neurons of QPPNet need to wait for the output results of the lower layer to run, and the tree convolution network's parameter scale is large, significantly impacting its efficiency. The tree-structured



(a) JOB-light (b) Synthetic (c) Scale (d) TPC-H (test)

(e) JOB-light (f) Synthetic (g) Scale (h) TPC-H (test)

Fig. 4: Ablation Study of BICE

model in QueryFormer contains many parameters, resulting in a high calculation cost. TPool extracts information from the plan based on a one-hot encoding method and adds bitmap, which results in ample space occupied by its encoding.

In BICE, the encoding is made as simple and efficient as possible by the *feature extractor*, the *compressor* reduces the learning difficulty of the estimation model, and the bidirectional LSTMs (i.e., $LSTM$ and $\widetilde{LSTM}$) can compute in parallel. Although we build an ensemble model composed of multiple models based on active learning and transfer learning, it only requires one additional training (because we extract $4 \times 25\% = 100\%$ of the data). Furthermore, only lightweight networks (i.e., three-layer fully connected neural networks) need to be updated when building the ensemble model. Thus the computational efficiency of BICE is comparable to some query statement-based models, with lower training and testing time than MSCN, QPPNet, TPool, and QueryFormer.

C. Ablation Study

In this section, we analyze the impact of critical components of BICE on the estimation accuracy, including the *feature extractor*, *compressor*, and an ensemble model.

Feature Extractor. As shown in Fig. 4, BICE-NoJoin, BICE-NoPre, and BICE-NoInfo represent our ablation studies on *feature extractor*. BICE-NoInfo that is BICE without the *information encoder*. BICE-NoJoin uses the join encoding method in MSCN (based on one-hot encoding) and without the *join encoder*. BICE-NoPre uses the filter encoding method in TPool without the *filter encoder*.

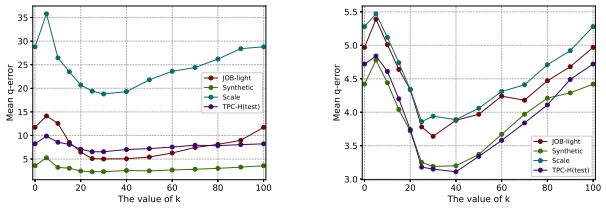
From the figures, we can see that BICE is significantly better than the other three models, BICE-NoJoin and BICE-NoPre performing worse because BICE-NoJoin does not consider the relationships between columns, and BICE-NoPre's encoding has the sparsity problem, which increases the learning difficulty. Therefore, the *join encoder*, *filter encoder*, and *information encoder* in BICE have a positive impact on the estimation accuracy.

Compressor. To study the effect of the *compressor* on BICE, we evaluated two variants: BICE-LSTM and BICE-LSTM2. BICE-LSTM that uses a single LSTM and without the *compressor*. BICE-LSTM2 uses two-layer unidirectional LSTMs without the *compressor*. As shown in Fig. 4, the accuracy of BICE-LSTM is the worst, and the efficiency of BICE-LSTM2 is lower than BICE (because the LSTM of the upper layer needs to wait for the output of the lower layer in BICE-LSTM2). Still, the estimation accuracy of BICE-LSTM2 is lower than BICE. Therefore, the bidirectional LSTMs-based compressor can effectively learn the information in the encoding.

Batch Training. In Section IV-A4, we propose two batch training methods, and the method adopted by BICE is *padding-based batch training*. BICE-SeqBatch that uses *sequence-based batch training* for batch training, and we can see from Fig. 4 its estimation accuracy is lower than BICE. The batch training method of BICE can adapt to a broader data range (because longer node sequences may appear in the test set), and the *compressor* can effectively avoid the knowledge-forgetting problem caused by padding.

TABLE III: Results of Different Data Sampling Strategies

	Mean q-errors (cardinality)			
	JOB-light	***Synthetic***	***Scale***	***TPC-H(test)***
BICE_strategy12	7.89	2.67	21.9	7.81
BICE_strategy13	7.23	2.53	22.7	7.21
BICE	**5.10**	**2.30**	**19.4**	**6.54**
	Mean q-errors (cost)			
	JOB-light	***Synthetic***	***Scale***	***TPC-H(test)***
BICE_strategy12	3.98	3.41	4.01	3.79
BICE_strategy13	4.02	3.39	3.92	3.24
BICE	**3.78**	**3.25**	**3.86**	**3.18**



(a) Cardinality errors (b) Cost errors

Fig. 5: Analysis of k

Ensemble Model. BICE-NoAl does not sample data through active learning and without building an ensemble model. As seen in Fig. 4, BICE-NoAl has the most significant estimation error due to its lack of learning ability for multiple data categories. Thus the ensemble model in BICE has a significant positive impact on the accuracy of the results. The value of the data sampling proportion k for active learning also affects the estimation accuracy, which we will analyze in the next section.

Analysis of Data Sampling. Fig. 5 shows the results of our analysis for the value of k. The model is equivalent to BICE-NoAl when $k = 0$ or $k = 100$. As can be seen from the figure, the error in the estimation results reaches a minimum when the value of k is between 20 and 40, and when $k < 20$ or $k > 40$, active learning plays a minor role. The data obtained for each category based on different sampling strategies become more similar, so the estimation accuracy decreases continuously. Tab. III shows the effects of using different data sampling strategies on the estimation results, BICE_strategy12 that uses *strategy 1 and 2* for data sampling, BICE_strategy13 that uses *strategy 1 and 3*, and we set their k to 50 for a fair comparison. From the table, we can see that the estimation errors of the two methods are significantly higher than BICE because although BICE_strategy12 selects data with training value, it selects fewer types of data. In BICE_strategy13, it selects data with a single indicator, making it difficult to measure the training value of the data accurately. Therefore, the sampling strategy designed in this paper positively impacts estimation accuracy.

V. Related work

Cardinality and cost estimation are the core tasks in query optimization, and accurate estimation results can effectively avoid producing poor execution plans. Recently, many estimation methods based on learning models have achieved better results than traditional ones. In this section, we provide an overview of the relevant models.

A. Data-driven estimation model

Naru [15] bases on an autoregressive model for learning data, which encodes discrete data using one-hot encoding or embedding, after which each row of data feed into the autoregressive model and training complete by minimizing cross-entropy loss. Naru computes by filters in queries and shows promising results on single table queries. NeuroCard [6] improves on Naru by proposing a weighted data selection method that draws training data from a joined table. Like Naru, it uses the cross-entropy function to calculate the loss and update the parameters in the autoregressive model. Compared with Naru, NeuroCard performs better on multi-table queries. Quicksel [49] is a cardinality estimation method based on the uniform mixture model. It extracts the ranges corresponding to each column from the query to construct a rectangle, removes many ranges from the queries, and uses quadratic programming for learning. Finally, the weighted summation uses to output the estimated result. Quicksel can efficiently perform calculations on multi-table queries. Data-driven models have been highly successful, with their estimates being more robust than other estimation methods, but have poor performance for scenarios with frequent data changes.

B. Query-driven estimation model

MSCN [8] designs a multi-set convolutional network based on Deep Set for cardinality estimation, which extracts tables, joins, and filters information from the query to input into different sets for calculation. It incorporates bitmap information to improve the accuracy of estimation. The estimation error of MSCN on multi-table queries is significantly lower than that of traditional estimation methods. Fauce [12] also extracts information from the query for encoding. For filter conditions, it uses data range representation for encoding. For join conditions, it uses an embedded method to learn the information of columns. Due to the complex situation of multiple tables joins in the query, it designs a multi-table join algorithm to encode joins. QPPNet [8] is a cost estimation model based on tree convolutional networks, which uses different neurons for different types of nodes in the query plan, and the neurons in the upper layer receive the information output from the lower layer. QPPNet finally outputs the estimated cost corresponding to the query plan. The query-driven estimation model has unique advantages in multi-table queries and is less affected by data changes. In addition, we can well add the estimation model based on the query plan tree to the query optimizer to find a better execution plan.

VI. Conclusion

We present a bidirectional compressor-based ensemble learning framework for efficient cardinality and cost estimation in query optimization. The framework encodes query plan trees into embeddings and employs a bidirectional LSTM-based compressor to obtain compressed embeddings. An ensemble model utilizing Bayesian neural networks, active learning, and transfer learning is incorporated. Empirical studies demonstrate enhanced accuracy and computational efficiency compared to the state of the art.

Acknowledgment

This work is partially supported by NSFC (No. 61972069, 61836007, 61832017, 62272086), Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021), Municipal Government of Quzhou under Grant No. 2022D037, and Key Laboratory of Data Intelligence and Cognitive Computing, Longhua District, Shenzhen.

REFERENCES

[1] Ibm knowledge center. [Online]. Available: www.ibm.com

[2] 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.

[3] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," in *SIGMOD '79*, 1979.

[4] 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.

[5] 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, 2021.

[6] Z. Yang, A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica, "Neurocard: One cardinality estimator for all tables," *Proc. VLDB Endow.*, vol. 14, pp. 61–73, 2020.

[7] J. Sun and G. Li, "An end-to-end learning-based cost estimator," *ArXiv*, vol. abs/1906.02560, 2019.

[8] 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, 2019.

[9] Z. Wu and A. Shaikhha, "Bayescard: A unified bayesian framework for cardinality estimation," *ArXiv*, vol. abs/2012.14743, 2020.

[10] R. Marcus and O. Papaemmanouil, "Plan-structured deep neural network models for query performance prediction," *ArXiv*, vol. abs/1902.00132, 2019.

[11] 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.

[12] 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.

[13] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, "Learning-based query performance modeling and prediction," *2012 IEEE 28th International Conference on Data Engineering*, pp. 390–401, 2012.

[14] X. Zhou, C. Chai, G. Li, and J. Sun, "Database meets artificial intelligence: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, pp. 1096–1116, 2022.

[15] 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.

[16] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das, "Deep learning models for selectivity estimation of multi-attribute queries," *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020.

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

[18] 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.

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

[20] Y. Zhao, G. Cong, J. Shi, and C. Miao, "Queryformer: A tree transformer model for query plan representation," *Proc. VLDB Endow.*, vol. 15, no. 8, p. 1658–1670, jun 2022. [Online]. Available: https://doi.org/10.14778/3529337.3529349

[21] H. Dombrovskaya, B. Novikov, and A. Bailliekova, *PostgreSQL Query Optimization*. Springer, 2021.

[22] P. Martins, P. Tomé, C. Wanzeller, F. Sá, and M. Abbasi, "Comparing oracle and postgresql, performance and optimization," in *World Conference on Information Systems and Technologies*. Springer, 2021, pp. 481–490.

[23] 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," *ArXiv*, vol. abs/2109.05877, 2021.

[24] C. M. Bishop, "Neural networks and their applications," *Review of scientific instruments*, vol. 65, no. 6, pp. 1803–1832, 1994.

[25] B. Momjian, "Explaining the postgres query optimizer," 2015.

[26] 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.

[27] R. E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, pp. 146–160, 1972.

[28] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl. Based Syst.*, vol. 151, pp. 78–94, 2018.

[29] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[30] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate club: An api oriented open-source python framework for unsupervised learning on graphs," *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020.

[31] L. Fröhlich, "Postgresql," 2022.

[32] 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.

[33] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, pp. 2451–2471, 2000.

[34] 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 *NIPS*, 2017.

[35] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, pp. 2673–2681, 1997.

[36] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *ArXiv*, vol. abs/1508.01991, 2015.

[37] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, 2018.

[38] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers of Computer Science*, vol. 14, pp. 241 – 258, 2019.

[39] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *ArXiv*, vol. abs/1505.05424, 2015.

[40] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *ArXiv*, vol. abs/1506.02142, 2016.

[41] E. Kaufmann, O. Cappé, and A. Garivier, "On bayesian upper confidence bounds for bandit problems," in *AISTATS*, 2012.

[42] F. Zhdanov, "Diverse mini-batch active learning," *ArXiv*, vol. abs/1901.05954, 2019.

[43] A. Kirsch, J. R. van Amersfoort, and Y. Gal, "Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning," in *NeurIPS*, 2019.

[44] S. Liu, X. Chen, Y. Zhao, J. Chen, R. Zhou, and K. Zheng, "Efficient learning with pseudo labels for query cost estimation," *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022.

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

[46] 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, 2021.

[47] M. Barata, J. Bernardino, and P. Furtado, "An overview of decision support benchmarks: Tpc-ds, tpc-h and ssb," *New Contributions in Information Systems and Technologies*, pp. 619–628, 2015.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[49] Y. Park, S. Zhong, and B. Mozafari, "Quicksel: Quick selectivity learning with mixture models," *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020.