Learned Index Benefits: Machine Learning Based Index Performance Estimation

Jiachen Shi
Nanyang Technological University, Institute for Infocomm Research, Singapore
jiachen001@e.ntu.edu.sg

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

Xiao-Li Li
Institute for Infocomm Research, A*STAR Centre for Frontier AI Research, Nanyang Technological University, Singapore
xlli@i2r.a-star.edu.sg

ABSTRACT

Index selection remains one of the most challenging problems in relational database management systems. To find an optimum index configuration for a workload, *accurately and efficiently quantifying the benefits of each candidate index configuration is indispensable.* As materializing each index configuration candidate and physically executing queries are infeasible, most of index tuners rely on the cost estimations from optimizer with "what-if" API. However, "what-if" based index benefit estimations have the following two limitations. Firstly, they generate significant errors, which compromise index recommendation quality. Secondly, generating query plans and benefit estimations for each candidate index configuration takes a considerable amount of time. To address the two challenges in index selection, we propose *an effective end-to-end machine learning based index benefit estimator.* In particular, we propose novel feature extraction and encoding techniques that do not rely on "what-if" call to generate query plan for each index configuration candidate. In addition, we design an attention mechanism to address index interaction issue and aggregate the impacts of different query operations. Finally, we leverage transfer learning technique to improve the estimator's learning ability for adaption to new database. Comprehensive experiments are conducted on different workloads, and extensive experimental results show that our proposed method outperforms "what-if" based index benefit estimations in terms of accuracy and efficiency. In addition, integrating our method into existing index selection algorithms can significantly improve index recommendation quality.

PVLDB Reference Format:
Jiachen Shi, Gao Cong, and Xiao-Li Li. Learned Index Benefits: Machine Learning Based Index Performance Estimation. PVLDB, 15(13): 3950 - 3962, 2022.
doi:10.14778/3565838.3565848

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/JC-Shi/Learned-Index-Benefits.

Proceedings of the VLDB Endowment, Vol. 15, No. 13 ISSN 2150-8097.
doi:10.14778/3565838.3565848

1 INTRODUCTION

For relational database management systems (RDBMS), creating an appropriate set of indices for a workload can boost the query performance by orders of magnitude. Nevertheless, the index selection problem (ISP), i.e., to find the optimal index configuration (i.e., a set of indices) for a workload, is an NP-Complete problem [31]. Solving ISP is challenging because of three reasons: (1) the number of potential index configurations is large when there are a large number of indexable attributes, and each index in a configuration can have multiple attributes with different orders, (2) the presence of an index mutually affects other indices' benefits on the workload; such interplay is called index interaction (IIA) [37], and (3) without materializing the indices and executing the queries, it is hard to quantify the impact of an index configuration. In real world applications, index selection is typically conducted by Database Administrators (DBAs). The expenses for DBAs become a significant factor in Total Cost of Ownership (TCO) [9]. To reduce TCO, automating index recommendation and implementation are needed. During the past decades, auto-indexing has been an active research area where many index selection algorithms [7, 11, 36] and commercial tools [1, 49] have been proposed to recommend the optimal index configuration.

For index selection problem, apart from the selection algorithm, accurately and efficiently quantifying the benefits of each index candidate on a query workload has a significant impact on the recommendation quality. To estimate the benefit of an index configuration on a query workload, most existing index selection algorithms rely on "what-if" calls which are based on database optimizer's cost model since actual query execution under all index candidate configurations is prohibitively expensive. A "what-if" call [8] is an index analysis utility which is supported by most DBMS. It creates hypothetical indices for index benefit estimations through providing the statistical information of index configurations to the database optimizer, and thus benefit estimation can be conducted without the actual creating or dropping of indices.

Although "what-if" based benefit estimation is widely adopted by many index tuning methods, there are two main drawbacks. First, "what-if" calls based on existing database optimizer may suffer from low accuracy in benefit estimation [20, 46]. During index selection, the benefit prediction errors may result in loss of potential improvement or even cause query regression (i.e., performance degradation) [3]. Although many solutions including the recent proposals [25, 26, 41] that use machine learning techniques have been proposed to improve the optimizer's cost model, they either does not consider or does not well capture the impact of indices.

Moreover, as the "what-if" based benefit estimation will be further used for index candidate comparison, training these machine learning (ML)-based cost models cannot directly minimize the error metric that corresponds to comparison errors [12].

Second, "what-if" calls take a considerable amount of time during index tuning. When index selection is invoked for a large workload with many index candidates, a large number of "what-if" calls are required to estimate benefits for different candidates. Using "what-if" calls will become a bottleneck of index selection [17]. According to [30], average 90% of the run-time of index tuning is spent on the "what-if" calls instead of selection logic.

To address the two drawbacks, this paper proposes a new machine learning based method to replace "what-if" calls for secondary index benefit estimation. *To the best of our knowledge, this is the first machine learning based method to quantify index benefit.* In particular, there are four key objectives to design an effective and efficient learning-based index benefit estimator:

- **Accuracy.** The index benefit estimation should be accurate so that the index tuner can differentiate different candidate index configurations correctly.
- **Generalization.** The learning-based method should be generalized to unseen queries as workload characteristics keep changing.
- **Efficiency.** Index benefit estimation using learning-based method should be efficient so that more candidates can be evaluated within the time window for index selection.
- **Adaptability.** The machine learning model should adapt to new dataset efficiently as recollecting training data for a new dataset is time-consuming.

To achieve these objectives, we propose an end-to-end machine learning based estimator **Learned Index Benefits** (LIB). In particular, the index benefit estimation problem is formulated as a regression task, where the target of the task is defined as the normalized cost difference after an index configuration being materialized. To extract and represent features from a query plan and an index configuration, we define a new concept called index optimizable operations and propose a novel featurization method that represents features as a set of index optimizable operations without using "what-if" call to generate query plan for each index configuration. Furthermore, for more accurate prediction, we propose to use attention based neural networks to learn the correlations between index optimizable operations and to deal with index interactions. In addition, when LIB is applied for a different databases, we leverage transfer learning [29] to reduce the demand of model re-training. Specifically, instead of training LIB from scratch, we use LIB that is trained on other datasets as a pre-trained model and fine tune the whole model with new data. Our experimental results show that the pre-trained model can outperform PostgreSQL and transfer learning will increase the convergence rate and reduce the needs for new data. Lastly, as LIB allows multiple candidates being evaluated in parallel, the efficiency of index selection is improved. We show that LIB can enhance the end-to-end index selection recommendation quality and greatly reduce the running time.

In summary, our major contributions can be summarized as:

- We propose an innovative machine learning based estimator LIB to replace "what-if" calls for index benefit estimation problem. *This is the first machine learning (ML)-based model that predicts index benefits without the need to generate query plans for each candidate index configuration* (Section 3).
- We define a *new concept called index optimizable operations* and present *a novel featurization technique* to represent a query under an index configuration as a set of index optimizable operations which does not rely on "what-if" calls (Section 4).
- We design an attention mechanism to deal with index interaction and improve LIB's estimation accuracy. In addition, LIB is able to adapt to different data schema with limited amount of new training data. (Section 5).
- We conduct comprehensive experiments to evaluate LIB's performance using different workloads. Experimental results show that LIB outperforms existing "what-if" based cost estimation, reducing up to 91% of prediction errors. Moreover, we demonstrate that integrating LIB into index tuner will improve index recommendations quality and reduce the end-to-end running time (Section 7).

2 PROBLEM STATEMENT

We first define the key concepts and notations used throughout this paper. Then we present our research problem statement.

Workload. A workload $\mathcal{W} = \{q_1, q_2, ..., q_n\}$ is a set of n queries on one or multiple tables in a relational database. A query q_j ($q_j \in \mathcal{W}, 1 \leq j \leq n$) is characterized by a set of columns in tables and a set of conditions for joining or filtering.

Secondary Index. Secondary index is a data structure that contains a subset of attributes from a table, with pointers that point to all the records which contain the specific key values of the attributes.

Index Configuration. An index configuration c_i is a set of indices. Each index may contain one or multiple columns from a table.

Index (configuration) Candidates. Index (configuration) candidate $C = \{c_1, c_2, ..., c_k\}$ is a set of k potential index configurations that are evaluated by index selection algorithms. Usually they are generated based on heuristic rules or the relevance to the queries.

Index Selection. Index selection (or index tuning) is about finding the best index configuration in C that enhances the query performance of database under some constraints, e.g., a limited storage budget or the runtime of the index selection [17].

During index selection process, the benefits of each index configuration quantify its impact on the workload for index candidate comparisons. In this paper, we define the benefits of an index configuration as follows:

Definition 1. **Index Benefits.** The index benefits of an index configuration c_i on a query q_j is defined as the query *execution cost reductions*, by comparing the execution cost when configuration c_i is materialized with the cost when no index is utilized. It is formulated as:

$$cr_{i,j} = Cost(q_j \mid \emptyset) - Cost(q_j \mid c_i) \tag{1}$$

where $Cost()$ is the execution cost of a query and $\emptyset$ represents the scenario that no index is utilized for execution.

Different from "what-if" based benefit estimation, we formulate the index benefit estimation problem as a regression task, to predict the cost reduction (or benefit) *directly* for a query under a specific index configuration. Intuitively, we propose to develop a learned cost model to estimate execution cost for each query under an index

configuration and compute the index benefits using the differences between two costs. However, it is challenging to build an accurate cost model since the query execution time may range from few milliseconds to thousands of seconds, depending on many factors[38], such as hardware, database size, conditions, etc. Moreover, based on [12], the error in a single cost estimation may translate to a poor index recommendation when the predicted cost value is used as performance metric for comparing candidate index configurations. Hence, estimating the execution cost for each query is not desirable for index benefit estimation. To provide a better metric to measure the benefit, we propose to estimate the **reduction ratio**. It is the fraction of running time being reduced by materializing an index configuration (i.e., the normalized cost reduction). In particular, the reduction ratio $rr_{i,j}$ is defined as:

$$rr_{i,j} = \frac{Cost(q_j \mid \emptyset) - Cost(q_j \mid c_i)}{Cost(q_j \mid \emptyset)} = \frac{cr_{i,j}}{Cost(q_j \mid \emptyset)} \quad (2)$$

With the reduction ratio as the target output, the constant query cost that no index is utilized ($Cost(q_j \mid \emptyset)$) can act as a reference for candidate comparisons. Hence, we are able to train an accurate index benefit estimation model LIB that is able to directly minimize the errors for comparing index configurations. This is a key design of our proposed solution.

Now, we define our research problem in this paper.

Problem Statement. *Index Performance (or Benefit) Estimation (IPE):* Given a workload $\mathcal{W} = \{q_1, q_2, ..., q_n\}$ with n queries and a set of index configuration candidates $C = \{c_1, c_2, ..., c_k\}$ with k index configurations, we aim to estimate the cost reduction ratio (normalized index benefits) $rr_{i,j}$ of each configuration c_i in C on each query q_j in the workload $\mathcal{W}$, where $i = 1, ..., k$ and $j = 1, ..., n$.

Taking the extracted features which represent a query q_j and the index configuration c_i as input, our proposed Learned Index Benefits (LIB) model estimates the cost reduction ratio which can be used for index selection. Particularly, we formalize the objective of our model as a *regret minimization problem*, where the regret for a query q_j with a specific index configuration c_i is defined as the squared residual (error) between the *actual* cost reduction ratio with the index configuration and the *estimated* one:

$$\underset{q_j \in \mathcal{W},\, c_i \in C}{\text{minimize}} \quad \mathcal{R}_{q_j|c_i} = (rr_{i,j(actual)} - \widehat{rr}_{i,j(estimated)})^2$$

By minimizing the sum of the squares of the residual between the actual and estimated reduction ratio, we aim to accurately estimate the index benefits.

3 LEARNING FRAMEWORK OVERVIEW

The end-to-end index benefit estimator LIB proposed in this paper contains three main components which are shown in Figure 1.

(1) **Feature Extractor** extracts useful features from a query and an index configuration, and represents them into a set of vectors (see Section 4), serving as input for *Encoder* component. Different from existing methods [12, 36] for index selection, we do not rely on "what-if" calls to generate query plan for each index candidate.

(2) **Encoder** aggregates the vector sets from extractor into a single vector for *Prediction model* component. We propose to leverage attention mechanism to learn the index interactions and the complex correlations between the feature vectors (see Section 5.1).

(3) **Prediction model** takes the representation vector from *Encoder* as input and outputs a reduction ratio to show the index benefits.

Workflow. As depicted in Figure 1, LIB operates in two phases: *offline* model training which is indicated by orange elements, as well as *online* estimation which is coloured as blue.

For **offline model training**, training data is collected through executing queries in workloads under different index configurations and encode into sets of vectors using *feature extractor*. The training data is stored as a tuple <vector representation, cost reduction ratio>. Then we train both the encoder and prediction model in an end-to-end supervised fashion. The offline model training can be conducted without any interruption to DBMS. The detail of model training is discussed in Section 5.3. To enhance the model's adaption to new databases, we carefully design feature attributes to be schema agnostic as much as possible and we propose to adopt a light-weight transfer learning strategy (see Section 5.4).

For **online estimation**, when an index tuner is invoked to recommend an index configuration for a workload, it will first generate a set of candidates. Then LIB is adopted to estimate the reduction ratios (i.e., quantified benefits) for each index candidate, instead of invoking "what-if" calls in existing index tuners to estimate execution cost. The outputs of LIB are used by the index selection algorithm to find the most beneficial index configuration directly (see Section 6). Note that LIB is independent of index selection algorithms, i.e., it can be integrated into different index tuners.

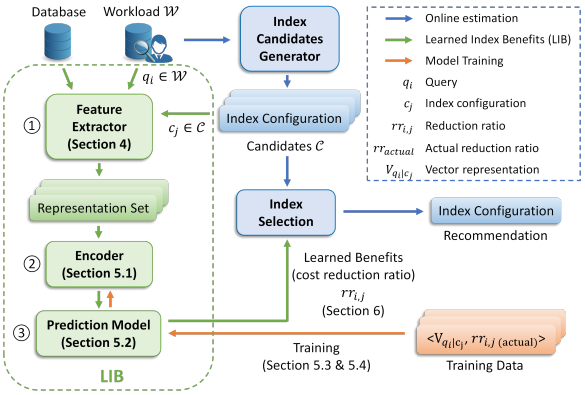


Figure 1: Overview of Learning-based Index Evaluation

4 FEATURE REPRESENTATION

Learning representation is one of the most important problems for machine learning algorithms to succeed [19]. The input for learned index performance estimation (IPE) task consists of a query and an index configuration. In this section, we introduce a novel featurization method to represent the input to learned IPE as a set of vectors without any "what-if" call. We explain the rationale of our proposed featurization in Section 4.1, and present details of the proposed feature representation methods for LIB in Section 4.2.

4.1 Rationale and Idea

IPE aims to estimate the cost reduction ratio of each index configuration on a query, where the cost of the query under the condition

that no index is utilized is used as a reference. To provide a representation of the input (i.e., a query and an index configuration) for LIB to learn the IPE task, we should featurize factors that contribute to the indices' cost reductions on the original query. There are two challenges in featurizing the input: First, off-the-shelf featurization methods in machine learning for database query-resource (e.g. execution cost) prediction [12, 25, 26, 41] rely on "what-if" calls to generate query plans for different index configurations. However, [17, 30] show that generating query plans using "what-if" calls is the bottleneck in index selection. Hence, we aim to extract features without relying on any "what-if" call. Second, existing featurization methods cannot well capture the IPE related information. We next discuss several existing methods. (a) The methods [25, 26] for estimating cost of query plans do not incorporate any index configuration information in featurization. They only encode the operation types (e.g. index scan or sequential scan) in the query plan without featurizing any detail of the index used, nor the indexed columns that are important for IPE. (b) The end2end method [41] featurizes the index in the query plan representations in the form of metadata. However, it focuses on learning the inherent tree structure of the plan and encodes extra features that are not directly related to IPE such as operations that will not be affected by index. We will show in Section 7.2 that such featurization method is difficult for models to learn well for IPE. (c) The featurization method used in [12] is designed to characterize the differences between two query plans. It does not contain any index information.

The above two challenges call for designing a new featurization method for IPE. To address the first challenge, we propose a novel idea to focus on the original query plan (i.e., without any index configuration) which will be changed by the index configurations, rather than invoking "what-if" call to generate plan for each index configuration as does previous work. We propose to featurize the original query plan together with the index configuration, which allows LIB, the value network as discussed later, to learn the reduction ratio on the plan in a supervised fashion. Below, we illustrate this idea through an example.

Example: Figure 2 shows an query in TPC-DS with an index configuration. Instead of generating a new query plan and studying the differences between the new query plan and the original plan, we focus on the original plan which is shown beside the query. To learn the reduction ratio on the original plan by the index configuration, we can featurize the information of how the query access (i.e., the sequential scan operations) and join (i.e., the hash join and nested loop operations) the data as well as the index configuration which can provide insights for models to learn its impacts.

To address the second challenge, we propose a novel featurization method to translate the original query plan with an index configuration into a set of vectors. According to the studies of the index functionality [28, 32], index optimizes an operation through changing the data retrieval method within that operation. The impacts of an index are at operation level. Moreover, the cost saving brought by an index solely depends on the scan range reduction, the impacts of an index are independent of the operation's position in the query plan. Based on these observations, we propose to featurize the query plan as a set of independent operations without incorporating any plan structural information, which are different from existing methods [25, 26, 41]. As not all the operations in

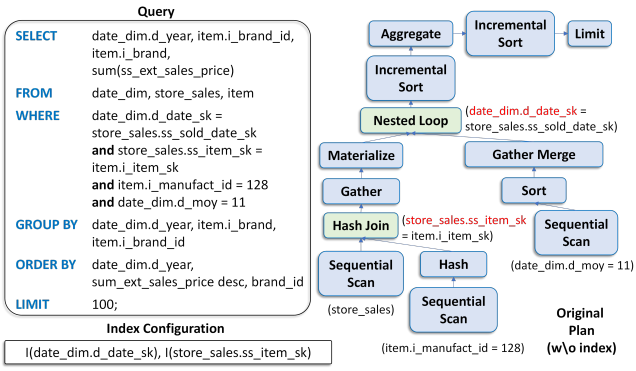


Figure 2: Example of Feature Extraction

a query plan will be affected by the index configuration, we can featurize only the portion of relevant operations. For this purpose, we define a new concept called *Index Optimizable Operations* O_{IO}:

Definition 2. **Index optimizable operations** O_{IO}**.** O_{IO} are operations in a query execution plan whose data source contains columns that are indexed by the index configuration. The O_{IO} is defined by three key information:

$$O_{IO} = [OI, DS, IC]$$

where O_{IO} is an index optimizable operation, OI, DS and IC are operation information, indexed column data statistics and index configuration information, respectively.

For the example in Figure 2, as the data sources of operations *Nested Loop* and *Hash Join* involve the indexed columns *"date_dim.d_date_sk"* and *"store_sales.ss_item_sk"*, they are identified as O_{IO}. We proceed to explain the three key information: OI, DS, IC.

- **Operation Information (OI):** The type of operation will determine how the operation being changed by the index. For instance, a join operation such as Hash Join may use index to retrieve join rows based on the selected rows from another table which replaces the usage of hash function. While for a scan operation, it may use index directly to access data based on predicates which replace the full table scan. To learn from the optimizer, we also leverage information generated by industrial-strength query optimizer such as estimated cardinality, which provide information regarding how many tuples being retrieved. Such information is critical for IPE as the smaller the portion of data being accessed, the higher the chance that the index is effective.
- **Database Statistics (DS).** The distribution and statistical information of the indexed column play a significant role in IPE. For example, the benefit of an index on a column is closely related to number of distinct values in the column. For columns with small distinct values, using index to retrieve data may not be effective as it still requires a significant amount of scanning.
- **Index Information(IC).** Different types of indices (single or multi-attribute) have diverse performance. For a multi-attribute index, beside encoding the type of the index, we may also encode the orders of the columns in the index because the effectiveness of the index is the highest at the leading (leftmost) column and decreases as the order increases.

In summary, we propose to featurize the original query plan with an index configuration as a set of *Index Optimizable Operations*, $\{O_{IO}\}$, for LIB to learn IPE. For the query and index configuration in Figure 2, we will featurize them as a set of two vectors where each vector of $\{O_{IO}\}$ captures the operation information (e.g., *Hash Join*), column statistics (e.g., *"store_sales.ss_item_sk"*) and index information (e.g., *I(store_sales.ss_item_sk)*).

4.2 Feature Extraction

We proceed to discuss the details of the three types of feature attributes for each O_{IO} and illustrate the proposed feature extraction process with an example.

(1) For **operation information**, we firstly classify the index optimizable operations into five types (join, sort, group, scan_range, scan_equal). Here, we further divide the scan operations into range scan and equal scan. This is because the scan operations with range predicate and with equal predicate will have different index scan ranges, and thus the respective effectiveness of an index is different. Each type of operations is encoded as a one-hot vector. In addition, we also use the estimated cardinality of each index optimizable operation as a feature dimension because it is relevant to index performance estimation. Logarithm transformation is used to reduce the skewness and variability of the estimated cardinality.

(2) For **database statistics**, we encode the number of rows, the NULL fraction and the ratio of distinct values of the indexed column which will affect the performance of an index. For NULL fraction and distinct ratio, their values are between 0 and 1, where 1 indicates all values are NULL/distinct and 0 represents the opposite. For number of rows, logarithm transformation is also applied.

(3) For **index information**, we use the type of index and the order of the indexed column in the multi-attribute index as feature attributes. We apply one-hot encoding to represent the type of index (single or multi-attribute). For a multi-attribute index, we use a feature with an integer value to indicate the order of an indexed column in that index. For example, for a multi-attribute index $I(A, B, C)$, the order of the indexed column B is 2.

In summary, the feature vector of each O_{IO} is in the form of $O^i_{IO} = [O_t, \log(card), \log(rows), dist\_frac, NULL\_frac, I_t, I_o] \in \mathbb{R}^{12}$, where O^i_{IO} is the i-th vector of $\{O_{IO}\}$, $O_t \in \mathbb{R}^5$ and $\log(card)$ represents operation type and estimated cardinality in operation information; $\log(rows)$, $dist\_frac$ and $NULL\_frac$ are the database statistics (number of rows, distinct fraction and NULL fraction), and $I_t \in \mathbb{R}^2$ and I_o featurize the index type and index order.

Example: Next, we illustrate the proposed feature extraction process using the example in Section 4.1. Figure 3 shows a graphical representation of this process. To represent a query with an index configuration as a set of vectors, we first generate the execution plan for the query under the condition that no index is utilized. For different candidate configurations on the same query, we can reuse the query plan generated previously. Then, based on the index configuration, we identify the indexed columns. Next, using these columns, we find out the index optimizable operations in the query plan. As defined above, we identify an operation as O_{IO} when its data source or condition contains the indexed columns. Based on Section 4.1, operations *Nested Loop* and *Hash Join* are O_{IO}. For each index configuration, we can find out all O_{IO} by one time scan of the query plan. After that we extract the database statistics from the database catalog (e.g. pg_stats in PostgreSQL [34]) for each indexed column. Lastly, we represent each feature dimension as discussed above and concatenate them to be a feature vector for each O_{IO}. For an index optimizable operation, there can be multiple indexes to optimize it. In this case, we represent them as separate feature vectors where each vector represents the impact of an index on the operation. The query and index configuration are then represented as a set of feature vectors, $\{O_{IO}\}$.

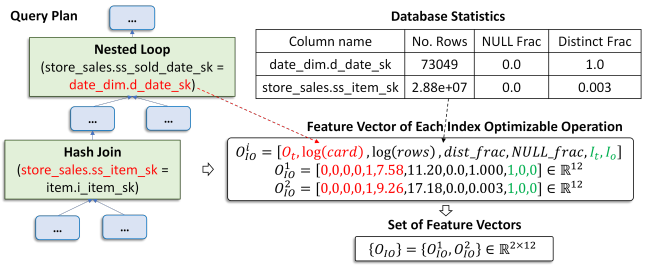


Figure 3: Example of Proposed Featurization Method

5 LEARNING INDEX BENEFIT ESTIMATION MODEL

In this section, we introduce the LIB, a deep neural network based value network for IPE. In particular, we present an aggregation encoder in Section 5.1 and the cost reduction ratio prediction model in Section 5.2. Finally, model training and adaption of LIB to unseen data are discussed in Section 5.3 and Section 5.4, respectively.

5.1 Encoder

To encode a set of vectors into a single vector for a typical regression algorithm, the encoder of LIB consists of three parts: embedding, learned representation and pooling as shown in Figure 4.

Embedding. The **embedding layer** converts the sparse vectors $O_{IO} \in \mathbb{R}^{12}$ from feature extractor to dense vectors $v_i \in \mathbb{R}^d$ where d is the embedding size. For each vector in the set of m vector representations $\{O_{IO}\} \in \mathbb{R}^{m\times12}$, the variability of the vector values is large. Hence, we use a one-layer fully connected neural network with ReLU activator to embed vectors from $\{O_{IO}\}$ into a set of compact latent vectors $\mathcal{S} = \{v_1, ..., v_m\} \in \mathbb{R}^{m\times d}$.

Learned representation and pooling. Accurately estimating the reduction ratio of an index configuration on a query is challenging in two aspects: (1) Index interactions (IIA) increases the complexity of IPE significantly as the interactions between indices heavily affect the overall benefits of the configuration. In fact, the significance of index interactions has been emphasized in several works on workload-driven index selection [4, 6, 36, 37]. Here, we use a simple example to illustrate the importance of IIA. Given an index configuration containing three indices I_1, I_2 and I_3, when I_1 and I_2 are used individually, their benefits may be trivial. However, when both I_1 and I_2 are employed together, they may provide a significant cost saving on the query due to index intersection. This is a positive interaction between I_1 and I_2. An example of negative interaction is as follows: Suppose I_3 has greater benefits when being used as

a substitution of I_1 and I_2; the existence of I_1 and I_2 may preclude the usage of I_3 during query execution. Thus, capturing these index interactions is crucial for models to better learn how the indices in a configuration affect the query performance. Nevertheless, existing method that characterizes and computes the degree of interactions between every pair of indices [37] requires significant number of "what-if" calls and it is not straightforward to encode all the interacting pairs with their degrees of interactions. (2) As the original query plan and the index configuration are featurized as a set of index optimizable operations $\{O_{IO}\}$ and the contribution of each O_{IO} to the overall cost reduction is not necessarily equal. Hence, it is difficult to aggregate the impacts of all O_{IO} to accurately estimate the reduction ratio.

To incorporate the complicate index interactions into LIB, we propose a **representation layer** which utilizes self-attention mechanism to model the IIA directly. Self-attention [45], also known as intra-attention, is a mechanism which explicitly models high-order interactions among the elements in a set. Several recent works have highlighted the competency of self-attention for set-input problems [14, 23]. To capture the complicate IIA information between each O_{IO}, we adopt the encoder block of the Transformer [45] without positional encoding to learn a representation for the $\{O_{IO}\}$. Given a set of latent vectors $\mathcal{S} = \{v_1, ..., v_m\} \in \mathbb{R}^{m \times d}$ from the embedding layer, we perform stacked self-attention encoding (SAE) to learn the representations. Each SAE can be formalized as follows:

$$\begin{aligned} SAE(\mathcal{S}) &= LayerNorm(H + FF(H)) \\ H &= LayerNorm(\mathcal{S} + Multihead(\mathcal{S}, \mathcal{S}, \mathcal{S})) \\ Multihead(\mathcal{S}, \mathcal{S}, \mathcal{S}) &= concat(O_1, ..., O_h)W^O \in \mathbb{R}^{m \times d} \\ O_j &= Att(\mathcal{S}W_j^Q, \mathcal{S}W_j^K, \mathcal{S}W_j^V) \end{aligned} \tag{3}$$

where FF is the feed forward neural network and Att is the attention encoding, which is proposed in [45]. Here, multi-head attention is adopted to expand LIB's ability to learn multiple relationships between the vectors. The encoder contains h sets of learnable parameters $\{W^Q, W^K, W^V\}_{j=1}^h$, where h is the number of heads, $W^Q, W^K, W^V \in \mathbb{R}^{d \times d^M}$ and $d^M = d/h$. The output from each head is later concatenated together to form the final representation. Residual connection [13] and layer normalization [2] are also applied in each SAE module.

To address the second challenge, we perform **pooling by multihead attention** (PMA) [19] to aggregate the set of leaned representations from SAE based on the inter-operation relations. It is shown to be beneficial for aggregation of set-input problem that has complicated interactions among elements. We first introduce a learnable vector $Z \in \mathbb{R}^{1 \times d}$, and then use one SAE to perform attention based aggregation with Z being the "query" and $SAE(\mathcal{S})$ being the "key" and "values". Lastly, a fixed dimension vectors $R \in \mathbb{R}^d$ will be outputted as final representation. The PMA is formalized as:

$$\begin{aligned} R &= LayerNorm(H' + FF(H')) \\ H' &= Multihead(Z, SAE(\mathcal{S}), SAE(\mathcal{S})) \in \mathbb{R}^{1 \times d} \end{aligned} \tag{4}$$

where $Multihead$ is the same as that in Equation 3.

Average pooling vs Attention. Alternatively, we can use fully-connected networks with average pooling similar to [16] without attention mechanism to encode the set of operations. However, we will show in Section 7, such a model may suffer from under-fitting.

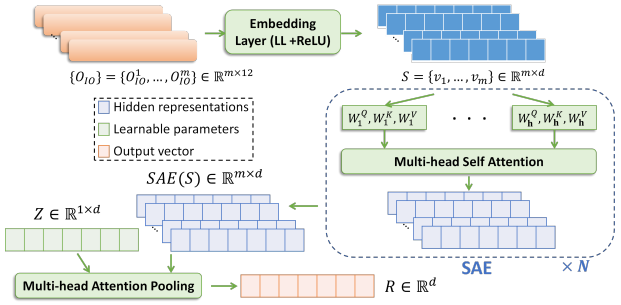


Figure 4: Overview of Encoder

5.2 Cost Reduction Prediction Model

Next we discuss the cost reduction ratio prediction layer. The prediction layer is a two-layer fully connected neural network with ReLU being the activator and Sigmoid function being the output layer. In LIB, the output cost reduction ratio is bounded between 0 and 1. For index configurations that cause query performance to degrade, we treat them the same as those that do not have any benefit so that the index tuner will be unlikely to select them. The output layer takes the vector representation R from the encoder as input and predicts the cost reduction ration $\widehat{rr}$.

5.3 Training Model

Training Data Generation. In LIB, we obtain training data by executing queries with real materialized index configurations and using the actual cost information as labels. First, given a workload, we adopt one of the index tuner algorithms (e.g., DTA[7], Extend[36]) to generate a set of index candidates for the workload. Then we create the index configurations using k-combinations [1] of the candidates set. For each index configuration, we materialize all indices and execute the workloads with the configuration. To minimize the impact from hardware, all queries are executed under warm-cache scenario (i.e., each query is executed 4 times and the average cost of the last three runs is reported as the execution cost). Next, we calculate the cost reduction ratio and adopt the feature extractor in LIB to encode the query and index configuration into a set of vector representations. Each training data is stored as a tuple <vector representation, cost reduction ratio>.

Training Overhead. Since LIB adopts the encoder block of transformer for feature encoding, the training time complexity of LIB grows quadratically with the number of index optimizable operations in each data (i.e., sequence length of the input) and grows linearly with the number of training data. Specifically, given a workload with n queries and m index configurations, there will be total $n \times m$ training data. For each data, let the number of index optimizable operations be k (all data is padded to the same number) and the feature size of each operation be d. The time complexity for LIB to evaluate the whole workload under all index configurations is $O(nmdk^2)$. Since k (maximum 40 in this paper) is relatively

[1] k-combination of a set is defined as a subset of k distinct elements.

smaller compared to number of all operations in a full query plan and $n, m \gg d, k$, we will show in Section 7.5 that the training time of LIB is manageable.

Generality. To improve the generality of LIB to unseen queries and prevent model overfitting, we adopt dropout [39] and early stopping regularization methods. Dropout with a probability (e.g., 0.2) is applied after each attention encoding layer and feed forward layer of SAE (Equation 3) and PMA (Equation 4). During model training, we monitor the model's performance on validation dataset at the end of every 20 training epochs. After training is finished, we callback to checkpoint with the highest validation accuracy.

5.4 Adaption

For machine learning models to perform well, sufficient amount of valid labelled training data is required. However, for index benefit estimation, collecting a large amount of data for model training is expensive and time-consuming. When the annotated training data is not enough, adaption of LIB to new data schema is challenging.

To enhance LIB adaption to new data schema, we firstly carefully design all feature attributes used in LIB to be database schema agnostic as much as possible which avoid LIB learning schema specific information. Besides that, we also propose to leverage transfer learning, which leverages knowledge learnt from a source domain with a large amount of training data to improve the learning performance in a target domain with limited data [29, 48]. It can reduce the demand of training data, increase the training speed and even improve the model performance.

When LIB is implemented for a new dataset with different data schema, instead of training the model from scratch using newly collected training data, we use the model trained by dataset with large amount of samples as a pre-trained model. Then we fine tune all pre-trained parameters in LIB for new data schema. Algorithm 1 shows the idea of transfer learning on LIB. Here, a lightweight transfer learning technique will meet our need because all feature attributes are schema agnostic, which allow LIB to learn the general and database schema independent mapping functions between the set of index optimizable operations $\{O_{IO}\}$ and the cost reduction ratio. For example, when an index is used to optimize a scan operation, the impact of the index solely depends on the change in scan range regardless of the content in the table.

6 INTEGRATION WITH INDEX TUNER

To recommend an index configuration to minimize the total execution cost of a given workload $\mathcal{W}$, an index tuner needs to perform index benefit estimation to assess the benefit of each index configuration on the workload to search for an ideal recommendation.

As the index benefit on each query outputted by LIB is normalized by the initial execution cost of the query and the execution costs of different queries are different, index tuner cannot directly use the cost reduction ratio for index tuning over all queries in the workload. To solve this issue, we multiply the estimated normalized ratios outputted from LIB with the initial cost [2] to covert the ratios into absolute cost reductions. The conversion is as follows:

$$\widehat{cr_{i,j}} = \widehat{rr}_{i,j} \times Cost(q_j \mid \emptyset) \tag{5}$$

[2]Here, we can use database optimizer or apply any existing cost estimator to get the cost prediction of each query under the condition that no index is involved.

Algorithm 1: Transfer Learning Algorithm

input : Training Dataset: $(\{O_{IO}\}_k, rr_k)_{k=1}^K$,
Pre-trained Parameter Set: Θ_0
output: Parameter Set for New Data Schema: Θ_1

$\Theta_1 \leftarrow \Theta_0$;
while *training is true* **do**
  $\mathcal{B} \leftarrow$ A Random minibatch of data;
  $\widehat{rr}_q \leftarrow \mathsf{LIB}(\{O_{IO}\}_q|\Theta_1) \quad \forall \{O_{IO}\}_q \in \mathcal{B},\ q = 1, ..., |\mathcal{B}|$;
  $\mathcal{L} \leftarrow \sum(rr_q - \widehat{rr}_q)^2/|\mathcal{B}| \quad \forall rr_q \in \mathcal{B}, q = 1, ..., |\mathcal{B}|$;
  for $\theta \in \Theta_1$ **do**
    $g_\theta \leftarrow \nabla_\theta \mathcal{L}(\Theta_1)$;
    $\theta \leftarrow \theta + \Gamma(g_\theta)$;
  end
end
return Θ_1;

where $cr_{i,j}$ is the estimated cost reduction of index configuration c_i on query q_j, $rr_{i,j}$ is the reduction ratios from LIB, and $Cost(q_j \mid \emptyset)$ is the initial cost of query when no index is utilized.

Based on the estimated cost reduction $cr_{i,j}$ of each candidate on the workload, the tuner is able to perform the index selection.

Parallelizable Property. Another desirable feature of LIB is that it is readily parallelizable. As index performance estimation using LIB involves only matrix multiplications and operations which can be parallelized, it can be significantly speeded up using devices such as graphics processing unit (GPU). Therefore, LIB allows the index tuner to evaluate multiple index candidates accurately in parallel.

7 EXPERIMENTS

We evaluate LIB's performance using widely used industry-standard benchmarks and real-world datasets on the following major facets:

- **Prediction accuracy**: we evaluate the accuracy of LIB for the tasks of estimating cost reduction ratios across queries and index configurations. We observe up to 91.2% reduction in the fraction of errors compared to PostgreSQL-13 optimizer's cost estimation. In addition, we also observe that LIB can reduce errors not only on average, but also across all error quantiles.
- **Improvement in index recommendations**: we integrate LIB into an index tuner and quantify the end-to-end improvement in workload execution cost. We show that augmenting the index tuner with LIB can enhance the index recommendation.
- **Ablation study:** We investigate the impacts of the proposed featurization method, the attention mechanism and the designed target output. We observe that these components are effective in accuracy improvement.
- **Efficiency**: We evaluate the running time of the online index benefit predictions and the end-to-end index tuning time for two index selection algorithms integrated with LIB on workloads with different sizes. We observe that LIB is more efficient than "what-if" based methods. Utilizing LIB can reduce up to 89% of the end-to-end running time for index tuning.
- **Adaption to different data schema**: We evaluate LIB's adaptability to different data schema. We find that LIB is effective in improving the model's learning performance.

Table 1: Statistics about the workloads

Benchmark	DB size (GB)	# Tables	# Queries	# Cases
TPC-H	10	8	700	3771
TPC-DS-10	10	24	390	27584
TPC-DS-50	50	24	198	4518
IMDB-JOB	9.3	20	113	2879

7.1 Experimental Setup

Evaluation metrics: To evaluate the prediction accuracy of LIB, following [24, 26, 41], we employ Q-Error as defined below:

$$Q\_Error = max(\frac{rr_{actual}}{\widehat{rr}_{predicted}}, \frac{\widehat{rr}_{predicted}}{rr_{actual}})$$

Workloads: To investigate LIB's performance on different workloads, we use four benchmarks: TPC-H-10GB [33], TPC-DS-10GB, TPC-DS-50GB [27] and IMDB-JOB. They have different characteristics in terms of number of potential indices, number of queries, and size of databases. For TPC-H we use a scale factors of 10 while for TPC-DS we use scale factors of 10 and 50. The queries on TPC benchmarks are generated based on the pre-defined templates using query generation tools [42, 43]. More specifically, for TPC-H, we use 14 templates to generate 700 queries. Following [17, 26], we exclude some templates whose execution costs are orders of magnitude higher than others. This is because they dominate the costs of the workloads and render the index selection problem less complex. An index speed up query from the excluded template would always outperform indices for other queries. For TPC-DS-10GB, due to the same reason, we use 78 out of 99 templates to generate 390 queries while we use 66 templates to generate 198 queries for TPC-DS-50GB. Internet Movie Data Base (IMDB) is real-world data. We use JOB Benchmark[3] which contains 113 queries.

Data generation: Using the method discussed in Section 5.3, we adopt the Anytime algorithm of the Database Engine Tuning Advisor (DTA) for Microsoft SQL Server [7] which is reproduced in [17] to generate index configuration candidates for workloads above. The details of the generated data are shown in Table 1, where # Case is the amount of instances generated. Each instance represents a query with an index configuration which is in the form of a tuple <vector representation, cost reduction ratio>. The number of tables in TPC-H benchmark is relatively small and the number of potential indices for each query is also limited. Therefore, the number of instances for TPC-H is small. For TPC-DS-50, as the actual execution time of each query is much larger compared to that of TPC-DS-10, smaller number of queries is adopted.

Following [12], we split the dataset into training and testing sets as follows. (1) **Index configuration:** we randomly split the union of all data points into five disjoint sets. Then we conduct 5-fold cross validation by taking 4 of them being the training sets and the remaining one being the test set. In this train-test splitting, index configurations in test set are different from those in training sets, so that the inference is performed on new index configuration that did not appear during training. (2) **Query:** the dataset is split into 5 disjoint sets based on the query information. We first split the queries into 5 disjoint sets, and then for each set of queries, we put the corresponding instances into one set. This simulates the setup where LIB is used for index benefit estimation on unseen queries. Through using these splittings methods, we simulate two types of variations in distributions between training and test sets.

Settings of Neural networks: As mentioned in Section 4, the data vector size is 12. The embedding size used in LIB is $d = 32$. For the encoder, 6 layers of 8 head self-attention modules are adopted with dropout rate equal to 0.2. The hidden dimension of the feed forward linear layer in encoder is designed as 128. The hidden dimension and output dimension for output layers are 64 and 1, respectively. The model is trained with Adam [15] optimizer using an initial learning rate of 0.001 for 150 epochs.

Methods: We compare LIB against the cost estimator from PostgreSQL 13.3 database as existing ISP algorithms [11, 17, 36] and index tuners are based on optimizers' cost estimations. We also compare with a state-of-the-art classifier-based solution "AI-Meet-AI" proposed in [12]. "AI-Meet-AI" utilizes classification algorithms to find the better plan between a pair of plans, but it does not support cost reduction ratio prediction. To extend "AI-Meet-AI" to estimate the cost reduction ratio, we replace the classifier component [12] with a two-layer fully connected neural network with hidden dimension 128 (the hyper-parameters are tuned by standard cross-validation) and Sigmoid output layer, and refer it as "AIMAI-R". Moreover, we also investigate a machine learning based cost estimator "end2end" [41] on IPE task. We follow the hyperparameters suggested in [41] and remove the string embedding part as string predicates are rarely related to secondary index. We train the model using both losses in cost and cardinality until the model converge. We also compare LIB with two operator level cost models (i.e., Plan-Structured Model (referred as Plan-Strut) [26] and MART [21]). We follow the hyperparameters as suggested in the papers to train their models. For MART, we adopt the default model without scaling function as there is less than 1% of test data with non-zero out_ratio (the model selection metric defined in [21]).

Environment: All evaluations are conducted on a machine with Intel i9-10900X CPU, 64GB RAM and GeForce RTX 3080.

7.2 Prediction Accuracy

The Accuracy of each method at estimating the cost reduction ratios of index configurations on queries is shown in Table 2. We repeat the experiments five times for testing on each set and the average results are reported.

The mean absolute error of the end2end model on TPC-DS-10 workload cost reduction ratio prediction is 32.637 which is several times larger than those of other methods. The reason for the poor performance could be: (1) based on [12], errors in single query cost prediction may result in significant errors when we compare two plans to calculate the cost reduction ratio. (2) The featurization method used in end2end method cannot well capture the difference of a query under diverse index configurations. For 61 (out of 78) queries in the test data set, end2end predicts the same execution cost for all different index configurations, leading to high errors in candidate comparisons. Hence, we do not include the end2end model as a baseline. Plan-Strut also performs worse than other baseline methods in prediction accuracy. It is challenging for models

[3] https://github.com/gregrahn/join-order-benchmark

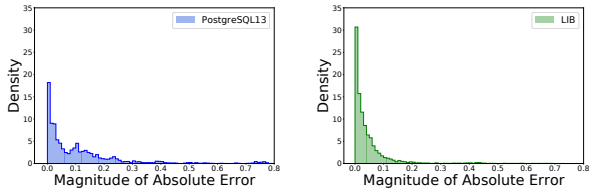
Table 2: Prediction accuracy (Q-Error) with different splitting methods

Dataset		Splitting by Configuration					Splitting by Query				
		PostgreSQL	AIMAI-R	Plan-Strut	MART	LIB	PostgreSQL	AIMAI-R	Plan-Strut	MART	LIB
TPC-DS-10	Mean	4.380	4.966	17.268	4.833	**1.984**	4.383	4.669	16.935	4.955	**2.205**
	90th	10.488	4.788	53.661	13.696	**3.260**	10.485	4.907	44.762	14.039	**3.935**
	95th	15.428	7.252	87.181	19.257	**5.170**	15.587	7.590	87.460	19.222	**6.340**
TPC-H	Mean	13.171	3.275	35.545	1.933	**1.159**	13.788	3.494	38.972	1.929	**1.148**
	90th	52.581	5.797	99.020	2.505	**1.324**	55.543	6.865	99.020	2.560	**1.310**
	95th	77.445	14.521	99.020	5.057	**1.575**	77.750	15.970	99.020	4.969	**1.562**
TPC-DS-50	Mean	6.603	7.499	33.195	15.263	**2.575**	6.439	4.476	33.479	15.219	**2.719**
	90th	16.623	8.524	93.195	52.150	**4.388**	15.374	8.309	89.063	49.917	**5.101**
	95th	38.507	12.841	99.020	61.235	**8.283**	33.775	14.512	97.827	59.853	**9.343**
IMDB-JOB	Mean	5.654	7.478	9.939	7.716	**3.655**	5.518	8.279	11.226	9.874	**4.363**
	90th	12.690	9.226	28.371	17.085	**6.904**	14.823	8.974	34.421	30.719	**8.031**
	95th	20.699	12.596	40.206	36.995	**9.513**	24.281	**10.396**	49.345	46.546	11.120

like end2end and Plan-Strut to differentiate query plans with only index configuration difference.

The results in Table 2 reveal that the proposed LIB model has the best performance among all methods. For inference on unseen configurations, LIB outperforms PostgreSQL, AIMAI-R, Plan-Strut and MART by up to 91.2%, 65.7%, 96.7% and 83.1% in term of mean Q-Error, respectively. This demonstrates that ML-based model using our proposed set-based featurization method and attention-based model structure is able to learn an accurate mapping function between index related information and cost reduction ratios. Although AIMAI-R outperforms PostgreSQL for TPC-H, it slightly underperforms PostgreSQL for other workloads. One reason for this observation is the featurization method proposed in [12] is designed mainly to identify the better plan in a pair, and it is difficult for AIMAI-R to learn the magnitude of the cost reduction between two plans. MART outperforms PostgreSQL and AIMAI-R on TPC-H but under-performs both methods for other workloads. We also investigate the performance of all methods at the tail (90th and 95th). As shown, LIB outperforms other methods across all error quantiles for TPC benchmarks and slightly (i.e., 0.724) underperforms AIMAI-R for 95 percentile on IMDB-JOB. On IMDB-JOB, although LIB under-performs AIMAI-R for 95 percentile, it outperforms AIMAI-R by 10.5% and 47.3% on 90 percentile and mean error respectively. Compared to PostgreSQL, LIB can reduce up to 98.0% errors in 95 percentile. This is a further evidence that LIB is able to estimate the reduction ratio accurately. On TPC-DS-50, the mean error of PostgreSQL is 6.60 which is 50% larger than that (4.38) on TPC-DS-10. This shows that the performance of PostgreSQL cost estimator is affected by the size of database. However, the mean error of LIB only increases from 1.98 to 2.57 and the 95th errors of LIB on TPC-DS-50 is better. This suggests that LIB is robust to changes in database size. One explanation for this observation is that LIB utilizes the database statistics as one of the feature attributes, allowing the model to capture the changes in database size and thus the performance is more stable.

For inference on unseen queries (the last 5 columns in Table 2), LIB's performance is similar to the results of inference on unseen configurations. The largest increase in mean error is only 0.708 on IMDB-JOB. This shows that LIB can generalize to unseen queries and it is robust to changes in query workload.



(a) PostgreSQL 13 **(b) LIB**

Figure 5: Prediction error distributions for TPC-DS

To further investigate the performance of LIB in cost reduction ratio estimation, we evaluate the absolute errors of LIB and PostgreSQL13. Figure 5 shows the absolute error distributions for PostgreSQL13 and LIB on the TPC-DS-10 workloads split by query. Each plot is converted into probability density where the area under the histogram integrates to 1. We omit plots for other cases as they follow the similar trends. As shown, LIB has the lower variance in error distribution and compared to PostgreSQL (Figure 5a), the error distributions for LIB shifts toward lower magnitude region. This shows that LIB can reduce prediction error across all error quantiles and it can estimate reduction ratio accurately for ISP.

7.3 Index Recommendation Quality

We now evaluate the impact of LIB on the end-to-end recommendation quality in terms of execution cost of queries when LIB is integrated into the index tuner. As discussed in Section 6, we integrate LIB into the index tuners that use the Anytime algorithm [7]. We conduct index tuning on 75 queries from TPC-DS-10GB, 66 queries from TPC-DS-50GB, 70 queries from TPC-H and 22 queries from IMDB-JOB which are excluded from the training data of LIB (i.e., LIB is implemented for index tuning on unseen queries). We then evaluate the performance of the tuner that use the same algorithm but with "what-if" based index benefit quantification methods which differentiate different candidates by PostgreSQL13's cost estimation and MART's prediction, respectively. Moreover, we compare LIB with "what-if" based classifier "AI-Meet-AI" proposed in [12]. We execute all candidates to find the optimal recommendation.

Table 3: TPCDS-10GB & TPCDS-50GB Workload level tuning (∗ denote methods based on "what-if" calls)

Methods	Total Cost Saving (ms)		Average Workload Improvement (%)		Distribution of Improvement (# workloads)										Total
					Regression		0%-5%		5%-20%		20%-50%		>50%		
	10GB	50GB	10GB	50GB	10GB	50GB	10GB	50GB	10GB	50GB	10GB	50GB	10GB	50GB	
PostgreSQL*	161,066	302,391	4.9	3.9	1	8	25	23	12	7	2	2	0	0	40
MART*	125,331	1,115,036	5.2	12.4	0	0	23	16	16	14	1	9	0	1	40
AI Meets AI*	1,488,759	425,664	8.1	5.0	0	0	27	31	11	6	0	3	2	0	40
LIB	3,071,577	1,175,872	15.5	12.5	0	0	24	14	8	16	2	9	6	1	40
Optimal	4,445,404	1,675,181	21.4	17.6	0	0	10	11	19	12	4	15	7	2	40

Table 4: TPCH-10GB & IMDB-JOB Workload level tuning (∗ denote methods based on "what-if" calls)

Methods	Total Cost Saving (ms)		Average Workload Improvement (%)		Distribution of Improvement (# workloads)										Total
					Regression		0%-5%		5%-20%		20%-50%		>50%		
	TPCH	IMDB	TPCH	IMDB	TPCH	IMDB	TPCH	IMDB	TPCH	IMDB	TPCH	IMDB	TPCH	IMDB	
PostgreSQL*	108,825	39,267	4.2	3.4	16	9	9	16	14	15	1	0	0	0	40
MART*	270,167	35,727	9.8	3.1	0	8	13	19	21	13	6	0	0	0	40
AI Meets AI*	176,930	19,578	6.5	1.6	0	12	26	23	10	5	4	0	0	0	40
LIB	277,209	42,321	10.0	3.6	0	8	12	15	22	17	6	0	0	0	40
Optimal	675,890	65,315	21.7	5.7	0	0	2	17	15	23	23	0	0	0	40

We aim to recommend the best index for a set of queries in a workload. We construct 40 query workloads (each contains 10 queries for TPC benchmarks and 8 queries for IMDB-JOB) by randomly sampling queries that are excluded from training data for each benchmark. Then we invoke index tuner using different index performance estimation methods to find the best index for each workload. The results of index recommendations are shown in Table 3 (TPC-DS) and Table 4 (TPC-H & IMDB-JOB). We report the total workload execution cost saving, average workload improvement and the distributions of improvements (i.e., the number of workloads in each improvement range).

For TPC-DS-10GB, the total cost saving of the LIB based index recommendations on the 40 workloads is $3,072s$ which is more than two times of that from "AI-Meet-AI". The LIB based index recommendation averagely improves a workload by 15% which is better than indices recommended by PostgreSQL13 and MART based tuners and "AI-Meet-AI". One reason that LIB based index recommendation has greater cost saving than "AI-Meet-AI" is LIB can quantify the benefits of each index configuration on the workload more accurately while "AI-Meet-AI" can only classify the better index configuration for each individual query, and "AI-Meet-AI" is mainly designed for query regression prevention. From the distribution of improvement, we observe that an optimal solution could achieve more than 50% improvement for 7 workloads. In contrast, "AI-Meet-AI" can only improve 2 of them by more than 50% while LIB can improve 6 of them. On TPC-DS-50GB, similar trends are observed, the total cost saving and average workload improvement of LIB ($1,176s$ and 12.5%) are more than two times of those from "AI-Meet-AI" ($426s$ and 5.0%) and PostgreSQL13 ($302s$ and 3.9%). When database size increases, PostgreSQL generates more query regression cases. However, both "AI-Meet-AI" and LIB are shown to be effective in preventing query regression. For MART, the index recommendation quality on TPC-DS-50GB is better than that on TPC-DS-10GB. One explanation for this is when database size increases, the execution cost differences between index configurations are larger. Although the prediction accuracy of MART is lower for TPC-DS-50GB, it is able to differentiate the query performance under various index configurations.

For TPC-H, as shown in Table 4, PostgreSQL generates more query regression cases (16). Nevertheless, "AI-Meet-AI" and LIB are effective in preventing them. For IMDB-JOB, because of complex attribute correlations and skew data distributions, it is more challenging to estimate cardinalities accurately. As shown in Table 4, all methods results in more query regression cases. However, LIB can still outperform other methods and achieve the largest cost saving.

7.4 Ablation Study

Featurization Method. To investigate the effectiveness of our proposed featurization method, we compare it against the featurization method proposed in [12]. In [12], they featurize a pair of plans into a vector by using a finite number of keys to represent the query operations and assigning values to each key to capture the differences of the operation between the two plans. To extend the featurization method for LIB, we first use "what-if" call to generate query plan for each index configuration, and then we form pairs of plan by comparing the plan for each index configuration with the original plan. Since the output vectors of [12] contain the values for all keys, we transform it into a set-structure data where each key is taken as an independent element in a set and its respective values (i.e., node cost and weighted bytes) are the feature attributes of that element. Table 5 shows that using our proposed featurization method can enhance the performance of LIB as LIB outperforms the model with featurization method in [12] ("AIMAI+LIB").

Attention Mechanism. To investigate the impact of the attention mechanism, we compare LIB against LIB without Attention (LIB-w/o attn) where the encoder in LIB is replaced with a fully connected deep neural network consisting of 1 embedding layer, 2 encoding linear layers with hidden dimension (32,256), an average

Table 5: Ablation Study - Prediction accuracy (Q-Error)

Dataset		LIB-Abs	AIMAI+LIB	LIB-w/o attn	LIB
TPC-DS-10	Mean	24.714	2.860	3.879	**2.205**
	90%	16.481	5.547	6.970	**3.935**
	95%	55.818	8.737	11.833	**6.340**
TPC-H	Mean	3.728	1.523	9.325	**1.148**
	90%	6.731	2.075	27.801	**1.310**
	95%	17.692	2.899	38.197	**1.562**
TPC-DS-50	Mean	39.315	3.418	4.312	**2.719**
	90%	37.108	5.354	9.522	**5.101**
	95%	135.621	9.951	16.871	**9.343**
IMDB	Mean	45.296	4.653	5.280	**4.363**
	90%	134.400	9.532	12.424	**8.031**
	95%	192.373	**10.177**	14.597	11.120

pooling mechanism and 3 linear output layers (hidden dimensions = (256,32,1)). In Table 5, the advantage of LIB over the LIB-w/o attn model on all datasets shows the usefulness of attention-based encoder and pooling mechanism. Intuitively, feature aggregation using attention is beneficial as it takes the influence of each instance on the target into consideration.

Prediction Output. To evaluate the effectiveness of using reduction ratio instead of absolute cost reduction as prediction output, we compare LIB against LIB with absolute cost reduction output (LIB-Abs) where we replace the Sigmoid output layer with an additional hidden layer and we apply logarithm transformation on the actual cost reduction label. As shown in Table 5, using reduction ratio significantly improves the performance of LIB.

7.5 Efficiency

Online Inference Time. We report the running time for online inference of LIB, AIMAI-R, MART and PostgreSQL for different numbers of IPEs during index tuning. Here, each IPE represents the cost reduction ratio estimation for an index configuration on a query in the workload. For LIB, MART and AIMAI-R, the inference time consists of the times taken for both data encoding and model evaluation. For PostgreSQL, we measure the time taken for "what-if" call to generate the execution plan. To ensure fair comparisons, all methods use the same hardware (i.e., without GPU) for inference. We vary the number of IPEs from 1 to 25,000, i.e., the number of estimations processed during index selection, and the results are shown in Figure 6. As LIB is readily parallelizable where it can process a batch of predictions simultaneously, LIB is more effective. As AIMAI-R and MART reliy on "what-if" call to generate query plan as input for each index configuration, their efficiency is limited by "what-if" call which provides further evidence that "what-if" call is the bottleneck for ISP algorithms. For 25,000 IPEs, LIB can reduce up to 97.8% of running time as compared to other methods.

Table 6: Training Time on Varying Training Sizes

Training Size (K)	5	7.5	10	12.5	15	17.5	20	22.5
Time (Minutes)	26	39	52	65	78	91	104	117

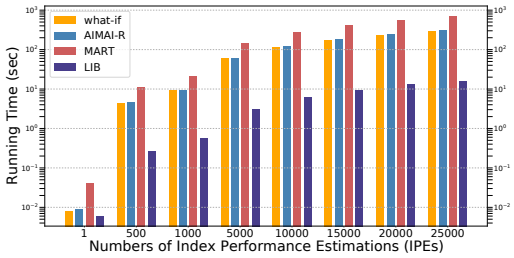


Figure 6: Online Inference Running Time

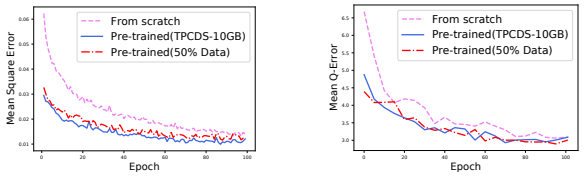
Offline Training Time. We show the training time on the TPC-DS-10 workload (maximum 36 index optimizable operations) with varying training sizes in Table 6. We observe that LIB takes 117 minutes to train with 22,500 data and the training time scales linearly with the size of the training data. Moreover, the training of LIB is done only once before integrating into index tuner. All training can be conducted *offline* externally such that it will not interrupt any online estimation process.

7.6 End-to-End Index Tuning Performance.

To evaluate the impacts of LIB on end-to-end index selection, we integrated LIB into two recent index selection algorithms (i.e., DTA [7] and EXTEND [36]) in lieu of original "what-if" based cost estimation methods and use them to conduct index selection to find the best index configurations for workloads from TPC-DS-10GB with sizes range from 10 to 3K. Meanwhile, we compare the performance with that of the same algorithms using "What-if" based cost estimations. All experiments are conducted under the same hardware environment (i.e., CPU) and use the same cache mechanism. The results are shown in Table 7. As shown, LIB is able to enhance the end-to-end index tuning quality. The recommended index configuration from tuner with LIB can greatly reduce the workload execution time. Furthermore, the algorithm running time is also reduced especially for workload with large size. For EXTEND, the LIB based algorithm's running time on workloads with sizes 10–300 is slightly larger. This is because the numbers of cost evaluations processed are much higher (698 and 30,466 vs 242 and 7,418). Overall, LIB is shown to be effective in enhancing index recommendation quality and index tuning efficiency.

Table 7: Results for Tuner Integrated with LIB

Algorithm		EXTEND		DTA	
Method	Workload Size	Running Time (s)	Workload Execution Time (ms)	Running Time (s)	Workload Execution Time (ms)
What-if	10	1.63	67,437	18.71	65,619
LIB	10	3.24	65,985	6.24	65,481
What-if	50	17.58	356,865	297.81	475,882
LIB	50	9.10	354,903	33.26	354,003
What-if	100	17.36	712,345	469.94	957,351
LIB	100	11.85	711,799	75.17	712,793
What-if	300	49.70	1,996,852	1,804.65	2,074,067
LIB	300	62.09	1,984,609	326.90	1,989,314
What-if	1K	169.89	7,071,291	5,112.69	9,882,302
LIB	1K	76.11	7,037,936	748.45	7,051,863
What-if	3K	461.97	21,206,376	18,034.33	29,788,098
LIB	3K	216.35	21,080,417	2,096.53	21,239,479



(a) TPCDS-50 Training **(b) TPCDS-50 Testing**

Figure 7: Learning curves of LIB

Table 8: Index Quality (Total Cost Saving (ms))

Model	TPC-DS-50	TPC-H	IMDB-JOB
Pre-trained LIB	310,773	104,639	41,652
PostgreSQL	302,391	108,825	39,268
Fine-tuned LIB	1,175,872	277,209	42,321

7.7 Adaption to Different Data Schema

We evaluate the effectiveness of the transfer learning technique. We train LIB under three different conditions: (1) training the model from scratch using only data from the new database, (2) pretraining the models with TPC-DS-10GB data and *fine tune* the whole model using data from the new database, and (3) same as condition 2 but *fine tune* the whole model using only half of the training data from the new database. The third condition is designed to evaluate the model's performance when number of training data is limited. For all LIB pretrainings, we train the model with 120 epoches.

For TPC-DS-50GB, the learning curves of LIB under the three conditions are shown in Figure 7. LIB, with transfer learning, converges faster than the model trained from scratch. Furthermore, the model with 50% of training data exhibits similar learning behaviours as the model with full training data. This suggests that utilizing transfer learning technique can largely reduce the need for training data and reduce the training time. Due to space constraints, we omit plots on other datasets, which show the similar trends.

Furthermore, we also evaluate the generalization ability of LIB to different workloads. We integrate the pre-trained model into the index tuner using the same settings in Section 7.3 and measure the index recommendation quality. Table 8 shows that without fine tuning, the pre-trained LIB models can outperforms PostgreSQL on TPC-DS-50GB and IMDB-JOB. This is largely because of our designed features. After fine tuning with thousands of data, LIB is able to outperform PostgreSQL on all datasets.

8 RELATED WORK

Index Selection. For the last several decades, automatic index tuning has been actively researched [10, 22, 40]. Many algorithms [7, 11, 36, 44] were proposed for efficient index selection. Recently, Kossmann et al. [17] compared and evaluated eight index selection algorithms. They created an open-source evaluation platform to facilitate performance analysis of different algorithms. In addition, there are ML-based approaches being proposed to learn an index advisor. Sadri et al. [35] and Lan et al. [18] proposed to used Deep Q-Learning to select the best index configuration.

Most of these index selection algorithms leverage the utility [8] "what-if" to create hypothetical index together with query optimizer's cost estimation for index tuning. However, "what-if" based index benefit estimation suffers from problems in terms of both efficiency and accuracy, as discussed in Introduction Section.

Several research works [5, 26, 41, 47] have been carried out to improve the database cost estimation accuracy. However, there are still significant errors when it is applied for index selection [12]. Recently, Ding et al. proposed to formulate index tuning as a classification task [12]. The main goal of the learned classifier is to enforce no query regression constraint and it works external to the optimizer. Different from our method that replaces the "what-if" call to improve the accuracy of index benefit estimation, it is invoked after the "what-if" based cost estimation to ensure no query regression. The learned classifier is orthogonal to our method and as shown in Section 7.3, modelling IPE as a regression task (i.e., using LIB) can achieve better index recommendations.

Featurization Methods. A great deal of featurization methods have recently been proposed in ML models on query performance prediction. In particular, operator-Table-Predicate representation has been adopted by many studies [24, 25, 41], where it first represents each operation node as a node vector, and then transforms the node vectors into a tree-structured vector. Although these representations have been proven to be useful, they either do not capture index information or cannot perform well for IPE as shown in Section 4.1. In [16], a set-based representation is used to represent a SQL query. However, it does not encode the physical operations nor index information. Hence, it is not suitable for IPE task.

Index Interaction (IIA). Several previous studies have proposed methodologies for modeling IIA. For instance, [4] used a heuristic approach to identify negative IIA. [37] proposed an efficient algorithm to compute the degree of interactions between indices. [36] modeled the IIA through constructing index selection in an iterative way. However, it is not straightforward to incorporate any of these methods into machine learning based value networks for IPE. Hence, we propose to design an attention mechanism to address IIA issue which achieves significantly better results.

9 CONCLUSION

In this paper, we proposed LIB, an end-to-end learning-based index benefit estimator, which is, to the best of our knowledge, the first machine learning based method to quantify index benefit for index selection. Particularly, we proposed a novel featurization method to encode the query with an index configuration as a set of index optimizable operations. In addition, a ML model with attention mechanism is proposed to address IIA and predict index reduction ratios accurately. Transfer learning technique is adopted to improve the model's adaptation capability to new data schema. Extensive experimental results on different benchmarks demonstrate that LIB can reduce up to 91.2% of errors in index benefit estimation. Lastly, integrating LIB into an index tuner can significantly improve the index quality and reduce the running time.

ACKNOWLEDGMENTS

This research is supported in part by MOE Tier-2 grants MOE2019-T2-2-181 and MOE-T2EP20221-0015.

REFERENCES

[1] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollár, Arunprasad P. Marathe, Vivek R. Narasayya, and Manoj Syamala. 2005. Database tuning advisor for microsoft SQL server 2005: demo. In *Proceedings International Conference on Management of Data, SIGMOD*. 930–932.

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* abs/1607.06450 (2016).

[3] Renata Borovica, Ioannis Alagiannis, and Anastasia Ailamaki. 2012. Automated physical designers: what you see is (not) what you get. In *Proceedings of the Fifth International Workshop on Testing Database Systems, DBTest*. 9.

[4] Nicolas Bruno and Surajit Chaudhuri. 2007. An Online Approach to Physical Design Tuning. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE*. IEEE Computer Society, 826–835.

[5] Surajit Chaudhuri. 2009. Query optimizers: time to rethink the contract?. In *Proceedings International Conference on Management of Data, SIGMOD*. 961–968.

[6] Surajit Chaudhuri, Mayur Datar, and Vivek R. Narasayya. 2004. Index Selection for Databases: A Hardness Study and a Principled Heuristic Solution. *IEEE Trans. Knowl. Data Eng.* 16, 11 (2004), 1313–1323.

[7] Surajit Chaudhuri and Vivek Narasayya. 2020. Anytime Algorithm of Database Tuning Advisor for Microsoft SQL Server. (June 2020).

[8] Surajit Chaudhuri and Vivek R. Narasayya. 1998. AutoAdmin 'What-if' Index Analysis Utility. In *Proceedings International Conference on Management of Data, SIGMOD*. 367–378.

[9] Surajit Chaudhuri and Gerhard Weikum. 2018. Self-Management Technology in Databases. In *Encyclopedia of Database Systems, Second Edition*. Springer.

[10] Sudipto Das, Miroslav Grbic, Igor Ilic, Isidora Jovandic, Andrija Jovanovic, Vivek R. Narasayya, Miodrag Radulovic, Maja Stikic, Gaoxiang Xu, and Surajit Chaudhuri. 2019. Automatically Indexing Millions of Databases in Microsoft Azure SQL Database. In *Proceedings International Conference on Management of Data, SIGMOD*. 666–679.

[11] Debabrata Dash, Neoklis Polyzotis, and Anastasia Ailamaki. 2011. CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads. *Proc. VLDB Endow.* 4, 6 (2011), 362–372.

[12] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2019. AI Meets AI: Leveraging Query Executions to Improve Index Recommendations. In *Proceedings International Conference on Management of Data, SIGMOD*. 1241–1258.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 770–778.

[14] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, S. M. Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. 2019. Attentive Neural Processes. In *7th International Conference on Learning Representations, ICLR*.

[15] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR*.

[16] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *9th Biennial Conference on Innovative Data Systems Research, CIDR*.

[17] Jan Kossmann, Stefan Halfpap, Marcel Jankrift, and Rainer Schlosser. 2020. Magic mirror in my hand, which is the best in the land? An Experimental Evaluation of Index Selection Algorithms. *Proc. VLDB Endow.* 13, 11 (2020), 2382–2395.

[18] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2020. An Index Advisor Using Deep Reinforcement Learning. In *The 29th ACM International Conference on Information and Knowledge Management, CIKM*. 2105–2108.

[19] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML (Proceedings of Machine Learning Research)*, Vol. 97. 3744–3753.

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

[21] Jiexing Li, Arnd Christian König, Vivek R. Narasayya, and Surajit Chaudhuri. 2012. Robust Estimation of Resource Consumption for SQL Queries using Statistical Techniques. *Proc. VLDB Endow.* 5, 11 (2012), 1555–1566.

[22] Sam Lightstone. 2018. Physical Database Design for Relational Databases. In *Encyclopedia of Database Systems, Second Edition*. Springer.

[23] Chih-Yao Ma, Asim Kadav, Iain Melvin, Zsolt Kira, Ghassan AlRegib, and Hans Peter Graf. 2018. Attend and Interact: Higher-Order Object Interactions for Video Understanding. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. Computer Vision Foundation / IEEE Computer Society, 6790–6800.

[24] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *Proceedings International Conference on Management of Data, SIGMOD*. 1275–1288.

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

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

[27] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The Making of TPC-DS. In *Proceedings of the 32nd International Conference on Very Large Data Bases*. 1049–1058.

[28] Benjamin Nevarez. 2016. *High Performance SQL Server: The Go Faster Book* (1st ed.). Apress, USA.

[29] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2010), 1345–1359.

[30] Stratos Papadomanolakis, Debabrata Dash, and Anastasia Ailamaki. 2007. Efficient Use of the Query Optimizer for Automated Physical Design. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07)*. 1093–1104.

[31] Gregory Piatetsky-Shapiro. 1983. The Optimal Selection of Secondary Indices is NP-Complete. *SIGMOD Rec.* 13, 2 (1983), 72–75.

[32] E. Pirozzi, I. Ahmed, and G. Smith. 2018. *PostgreSQL 10 High Performance: Expert Techniques for Query Optimization, High Availability, and Efficient Database Maintenance*. Packt Publishing.

[33] Meikel Pöss and Chris Floyd. 2000. New TPC Benchmarks for Decision Support and Web Commerce. *SIGMOD Rec.* 29, 4 (2000), 64–71.

[34] PostgreSQL 2021. *PostgreSQL 13.4 Documentation, chapter 51.88*. Retrieved Aug 26, 2021 from https://www.postgresql.org/docs/13/view-pg-stats.html

[35] Zahra Sadri, Le Gruenwald, and Eleazar Leal. 2020. Online Index Selection Using Deep Reinforcement Learning for a Cluster Database. In *36th IEEE International Conference on Data Engineering Workshops, ICDE*. 158–161.

[36] Rainer Schlosser, Jan Kossmann, and Martin Boissier. 2019. Efficient Scalable Multi-attribute Index Selection Using Recursive Strategies. In *35th IEEE International Conference on Data Engineering, ICDE*. 1238–1249.

[37] Karl Schnaitter, Neoklis Polyzotis, and Lise Getoor. 2009. Index Interactions in Physical Design Tuning: Modeling, Analysis, and Applications. *Proc. VLDB Endow.* 2, 1 (2009), 1234–1245.

[38] Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko. 2012. *High Performance MySQL: Optimization, Backups, and Replication* (3rd ed.). O'Reilly Media, Inc.

[39] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

[40] Michael Stonebraker. 1974. The choice of partial inversions and combined indices. *Int. J. Parallel Program.* 3, 2 (1974), 167–188.

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

[42] TPC-DS Query Generation 2020. *TPCDS-kit*. Retrieved Dec 1, 2021 from https://github.com/gregrahn/tpcds-kit

[43] TPC-H Query Generation 2018. *TPCH-kit*. Retrieved Dec 1, 2021 from https://github.com/gregrahn/tpch-kit

[44] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. 2000. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In *Proceedings of the 16th International Conference on Data Engineering*. 101–110.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Annual Conference on Neural Information Processing Systems*. 5998–6008.

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

[47] Wentao Wu, Jeffrey F. Naughton, and Harneet Singh. 2016. Sampling-Based Query Re-Optimization. In *Proceedings International Conference on Management of Data, SIGMOD*. 1721–1736.

[48] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2021. A Comprehensive Survey on Transfer Learning. *Proc. IEEE* 109, 1 (2021), 43–76.

[49] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam J. Storm, Christian Garcia-Arellano, and Scott Fadden. 2004. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*. 1087–1097.