PRICE: A Pretrained Model for Cross-Database Cardinality Estimation

Tianjing Zeng[1], Junwei Lan[1,2,#], Jiahong Ma[1,3,#], Wenqing Wei[1,2], Rong Zhu[1,*], Pengfei Li[1], Bolin Ding[1,*], Defu Lian[2], Zhewei Wei[3], Jingren Zhou[1,*]
[1]Alibaba Group, [2]University of Science and Technology of China, [3]Renmin University of China

ABSTRACT

Cardinality estimation (CardEst) is essential for optimizing query execution plans. Recent ML-based CardEst methods achieve high accuracy but face deployment challenges due to high preparation costs and lack of transferability across databases. In this paper, we propose PRICE, a PRetrained multI-table CardEst model, which addresses these limitations. PRICE takes low-level but transferable features w.r.t. data distributions and query information and elegantly applies self-attention models to learn meta-knowledge to compute cardinality in any database. M5 It is generally and adaptively applicable to any unseen new database to attain high estimation accuracy, while its preparation cost is as little as the basic one-dimensional histogram-based CardEst methods. Moreover, PRICE can be finetuned to further enhance its performance on any specific database. We pretrained PRICE using 30 diverse datasets, completing the process in about 5 hours with a resulting model size of only about 40MB. Evaluations show that PRICE consistently outperforms existing methods, achieving the highest estimation accuracy on several unseen databases and generating faster execution plans with lower overhead. After finetuning with a small volume of database-specific queries, PRICE could even find plans that were very close to the optimal ones. Meanwhile, PRICE is generally applicable to different settings such as data updates, data scaling, and query workload shifts.

1 INTRODUCTION

Cardinality estimation (CardEst), which predicts the number of tuples of each SQL query before execution, plays a crucial role in query optimization in DBMS. Accurate CardEst could significantly help to identify highly efficient execution plans, thus enhancing the performance of DBMS. As a result, CardEst methods have been extensively studied in the literature, but until now, their performance is still far from satisfactory [29].

Background and Challenges. Traditional methods, including one-dimensional (1-D) histograms [64, 66] and sampling [32, 45], are widely applied in commercial and open-source DBMSs [21, 51, 65, 67]. They only require gathering statistical features, i.e., the 1-D histogram for each attribute, and thus, they are friendly to deploy for any new database. However, their reliance on oversimplified models and unrealistic assumptions on data distributions often leads to poor estimation quality [82, 87]. To address these limitations, a surge in ML-based methods has been proposed in recent years. They either directly learn to model the joint probability density function (PDF) of data to compute cardinalities (data-driven methods [71, 87]) or build models mapping featurized queries to their cardinalities using collected query workload (query-driven methods [22, 23]) or both (hybrid methods [47]). Although existing benchmarks and evaluations [29, 39] have verified that these ML-based methods could attain high accuracy to find better execution plans, their explicit shortcomings still prevent them from being truly applicable:

1) These methods are all tailored to specific datasets and not R1D1 transferable across different databases. Data-driven approaches model the specific underlying data distribution for each dataset. Query-driven approaches typically represent features (e.g., the values of attributes or table names) as hard-encoded vectors, which can not be resolved in other datasets.

2) The preparation cost of existing ML-based methods is much higher than traditional CardEst methods. For each new database, they are required to collect training data, i.e., the sampled tuples in data-driven methods and the executed query workload in query-driven methods, tune hyper-parameters, train the models, and maintain the models for inference, which is time and space-consuming.

3) The performance of these ML-based methods is not stable [29, 39]. Data-driven methods often have prior assumptions on data distributions (e.g., DeepDB [36] assumes attributes are not highly correlated), while query-driven models can not perform well on queries having distinct distributions with the training workload [87]. Some methods [29, 39] can not well adapt to dynamic settings.

In light of these issues, what we desire is a CardEst method that could be effortlessly deployed on any unseen new database with minimal preparations (akin to traditional methods) while still maintaining high but stable accuracy comparable to ML-based methods.

Motivations and Our Contributions. We note that there has been some exploration into developing general models for cost estimation [15, 33–35] or single-table CardEst [53], but our goal is to resolve the most practical, but also complex, CardEst on multi-table join queries. We draw some inspiration from the success of pretrained models in the NLP domain, such as BERT [20] and ChatGPT [16]. They organize information hierarchically from the basic embedding of words (a.k.a. tokens) to complex semantic representations to tackle numerous NLP tasks simultaneously. In this paper, we lead the pretrained model into multi-table CardEst and propose PRICE, a PRetrained multI-table CardEst model that is universally appliable to any database.

Similar to NLP pretrained models, our PRICE also apply low-level but transferable features to learn high-level intermediate representations for CardEst. From a statistical view, the cardinality of any SQL query could be obtained by fetching the probability of the range specified by its filtering predicates on the joint PDFs of attributes across multiple tables. Therefore, the key tasks are selecting proper features, designing mechanisms to represent the joint PDFs, and fetching the probability by filtering predicates.

∗ Corresponding authors, primary contact email address red.zr@alibaba-inc.com
# The two authors contribute equally to this paper, listed in alphabetic order.

To this end, we leverage simple features in PRICE, including the value distribution of each attribute (represented as 1-D histogram), the distribution of scaling factors of each join condition $T \bowtie S$ (i.e., how many tuples in S are joined for each value in T, also represented as 1-D histogram) and some characteristics of each filtering predicate, as basic tokens in our pretrained model. Notably, the feature construction time of our PRICE is as low as the traditional 1-D histogram-based methods. Meanwhile, unlike existing CardEst methods, which are learned to fit each specific database and/or query workload, the feature representations in PRICE maintain the same semantics across different databases.

M1 M5 R1W1 R2W1 Central to our approach is a powerful self-attention mechanism that captures the meta-knowledge necessary for accurate CardEst. This mechanism adaptively assigns weights to tables and attributes based on their relevance to the final cardinality, providing crucial insights into token significance. During the fusion of multiple tables, PRICE adjusts its attention weights according to the distribution of the joined tables, ensuring a nuanced representation of the joint PDFs. This adaptability enables PRICE to effectively simulate the cardinality estimation process by condensing high-level embeddings that reflect both the backbone information of table joins and the intricate details required for filtering probabilities. Consequently, the final cardinality is accurately derived from these embeddings, making PRICE broadly applicable across various databases with minimal adjustments.

Due to the transferable features and attention mechanisms in PRICE, when it is trained over different databases with diversified joint PDFs and filtering conditions, it could be generally and adaptively applied to any database with reasonable and stable performance. To train PRICE, we gather a broad and varied collection of 30 datasets from various domains, generate 5×10^4 training queries on each of them, and collect their true cardinality for model training. This collection could serve as a new comprehensive benchmark for CardEst. On a common machine, our pretraining consumes only around 5 hours, which results in a model with a size of around 40MB. Moreover, the model could be finetuned over each specific database to improve its accuracy further.

By comprehensive evaluations in actual DBMS (PostgreSQL), we find that PRICE significantly outperforms other CardEst methods. The end-to-end execution time of the query plans generated by PRICE is consistently better than the original PostgreSQL and comparable to the advanced ML-based methods. After finetuning, the performance of PRICE is even close to the optimal plans. Meanwhile, we also find that PRICE is generally applicable to any new databases and different settings, i.e., data updates, data scaling, and query workload shifts.

In summary, our main contributions are listed as follows:

1) We design PRICE, a pretrained multi-table CardEst model that can be easily deployed on any new database with minimal preparations while preserving stable and high estimation accuracy. (in Sections 3 and 4)

2) We collect a new comprehensive benchmark for CardEst and carefully train PRICE to be applicable. (in Section 5)

3) We conducted extensive experiments to evaluate the performance of PRICE in various settings, confirming its effectiveness, generality, and robustness. (in Section 6)

2 PRELIMINARIES

In this section, we formalize the CardEst problem and review current CardEst methods. Based on this, we summarize some key findings that inspire our work.

CardEst Problem. A database D comprises a set of tables $T = \{T_1, \dots, T_N\}$ where each table T_i contains n_i attributes as $T_i = (A_{i,1}, \dots, A_{i,n_i})$. In this paper, we assume that each attribute $A_{i,j}$ is either categorical or continuous, where a categorical attribute falls into a finite domain $\text{Dom}(A_{i,j}) = \{c^1_{i,j}, \dots, c^d_{i,j}\}$ and a continuous attribute spans in an interval $\text{Dom}(A_{i,j}) = [\min_{i,j}, \max_{i,j}]$.

Given a SQL query Q on database D, we seek to estimate the cardinality $\text{Card}(Q)$—the exact number of tuples by executing Q on D—without actual execution. Let Q be the most common select-project-join (SPJ) SQL queries on table subset $T_Q \subseteq T$ with a set of join conditions J and filtering predicates F, represented as:

$$\texttt{SELECT COUNT}(*) \texttt{ FROM } T_Q \texttt{ WHERE } J \texttt{ AND } F. \quad (1)$$

In this paper, we formalize each join condition in J as $T_i.A_{i,x} = T_j.A_{j,y}$, which could either be a primary-foreign key (PK-FK) join or a foreign-foreign key (FK-FK) join. We represent each filtering predicate as "$T_i.A_{i,j}$ OP VALUE", where OP $\in \{<, \geq, >, \leq, =\}$ is a comparison operator and VALUE is picked from $\text{Dom}(A_{i,j})$. We reserve the support for other types of join, e.g., non-equal and non-inner join, and LIKE queries on string attributes for future work, since they are not the main focus of the CardEst problem [29, 47].

To formalize the CardEst problem, we consider it from the statistical perspective. R1D2 For each SPJ query Q in Eq. (1), in no ambiguity, we also use T_Q to denote the joined table accessed by Q. Let A_Q be the set of all attributes in T_Q. We can regard each attribute $A_{i,j}$ as a random variable defined over its domain $\text{Dom}(A_{i,j})$. Then, the table T_Q forms a joint probability density function (PDF) $\Pr(T_Q) = \Pr(A_{1,1}, A_{1,2}, \dots, A_{i,j}, \dots)$ on the domain $\text{Dom}(A_Q) = A_{1,1} \times A_{1,2} \cdots \times A_{i,j} \times \cdots$. Each record $t \in T_Q$ can be regarded as an independent sample drawn from $\text{Dom}(A_Q)$ by $\Pr(T_Q)$. After that, the query Q can also be represented in a canonical form as

$$Q = \{A_{1,1} \in R_{1,1} \wedge A_{1,2} \in R_{1,2} \wedge \cdots \wedge A_{i,j} \in R_{i,j} \wedge \dots\}, \quad (2)$$

where $R_{i,j} \subseteq \text{Dom}(A_{i,j})$ defines the constraint region specified by the filter predicate on attribute $A_{i,j}$ and $R_{i,j} = \text{Dom}(A_{i,j})$ if Q has no constraint on $A_{i,j}$. Then, the probability of a randomly picked record $t \in T_Q$ satisfying query Q is:

$$\Pr(Q) = \Pr(A_{1,1} \in R_{1,1}, A_{1,2} \in R_{1,2}, \dots, A_{i,j} \in R_{i,j}, \dots). \quad (3)$$

Obviously, when $|T_Q|$ hosts a sufficient number of tuples, we have

$$\text{Card}(Q) = \Pr(Q) \cdot |T_Q|. \quad (4)$$

Analysis of Existing Methods. We summarize existing CardEst methods developed under different technical paradigms. The details are reviewed as follows:

Traditional Methods, including histogram [19, 27, 28, 59, 64, 66, 75] and sampling [32, 38, 45, 46, 85], are widely applied in commercial and open-source DBMSs [26, 54, 62, 65]. They are based on simplified assumptions and expert-designed heuristics and only require gathering very simple statistical features, i.e., the 1-D histogram for each attribute. Hence, these methods are friendly to deploy for any new database, but their reliance on simplified models often leads to poor estimation quality [82, 87].

R1D2 *Query-driven Methods* attempt to learn the direct mappings function $\mathcal{F} : Q \rightarrow \text{Card}(Q)$. These methods require the collection of training queries and their corresponding true cardinalities, which usually take a long time. Classic methods apply queries to correct and tune histograms [17, 24, 37, 68] or update statistical summaries [69, 75]. Recent methods leverage advanced ML models, including DNNs [42, 43, 48, 49, 84], auto-regression [79], KDE [38], gradient boosted trees [22, 23], etc., to model the complex distributions and improve the estimation accuracy.

Data-driven Methods are independent of queries. R1D2 They directly model $\Pr(T_Q)$ and compute $\Pr(Q)$ from $\Pr(T_Q)$ to obtain $\text{Card}(Q)$ by Eq. (4). These methods involve learning unsupervised models of the joint distribution of multiple attributes and compute the cardinality using the models. A variety of ML-based models have been used in existing work, including the deep auto-regression model [30, 81, 82] and probabilistic graphical models (PGMs) such as Bayesian networks (BN) [18, 25, 71, 78], SPN [36, 63], and FSPN [80, 87], to model the joint data distribution.

Later, *Hybrid Methods* [23, 41, 47, 60, 76] further equip the function $\mathcal{F}$ with additional features on attribute distributions, e.g., the R1D2 histogram of $\Pr(A_{i,j})$ for each attribute in [47], so that it could better capture the data distribution information of $\Pr(T_Q)$ and exhibit better results than purely query-driven methods [84] according to the benchmark evaluations [29, 73].

While ML-based methods achieve high estimation accuracy, their lack of cross-database applicability limits their practical deployment. Our goal is to propose a method that is easily deployable on any unseen database with minimal preparation—merely requiring the gathering of simple statistical information similar to traditional methods—while preserving high estimation accuracy comparable to ML-based approaches. In the following, we present our intuitive roadmap in Section 3 and discuss its technical details in Section 4.

M1 M5

3 OUR ROADMAP

To lay the foundations of our method, we exhibit the similarities between NLP tasks and our CardEst problem. Based on this, we derive the key information to be captured in the CardEst pretrained model, as well as the suitable tools to acquire such knowledge.

Similarities between NLP Tasks and CardEst Problem. We R1W1 note that, our CardEst problem resembles NLP tasks. In NLP tasks, each word w is regarded as the fundamental data unit. Different combinations of words create texts with distinct semantics. To process them, we apply embedding methods, such as CBOW or Skip-gram [56], to map each w into a vector (a.k.a. token) in the latent semantic space. The embedding vector captures not only the frequency information of word w, but also its semantic joint relations with other words, i.e. , word co-occurrence in the contextual window. Based on such embeddings, the pretrained NLP models are developed to organize semantic information in a hierarchical manner. They start from low-dimensional word embeddings (tokens) and learn to capture complex semantics of phrases, sentences, and entire documents through extensive training on a large corpus. Such high-level representations are then applied to complex downstream tasks, such as text summarization and translation. Notably, for different word embeddings and NLP tasks, the pretrained models could apply different coefficients (derived from the learned parameters) to obtain distinct semantic representations. However, the models' framework always remains consistent. As a result, the pretrained NLP models attain transferability to different texts and NLP tasks.

Similarly, for relational tables, we could regard each attribute $A_{i,j}$ as the fundamental data unit (comparable to words). For the CardEst problem, the cardinality of query Q varies as the attributes result in different joint PDFs $\Pr(T_Q)$. We find that there also exists a hierarchical relation from $A_{i,j}$ to $\Pr(T_Q)$. Specifically, on each single table T_i, all attributes are correlated to form the joint PDF $\Pr(T_i)$ (analogous to phrases); and multiple tables are connected (through join conditions on attributes) to form the joint PDF $\Pr(T_Q)$ on the joined table (analogous to sentences). Therefore, it is also possible to represent the high-order PDF $\Pr(T_Q)$ using the combination of low-level attribute information. For example, for two attributes (or corresponding random variables) X and Y, the joint probability $\Pr(X = x_0, Y = y_0)$ of any point (x_0, y_0) could be approximated by combining the probability on each attribute and adjusted by coefficients, i.e. , $\Pr(X = x_0, Y = y_0) = \alpha \Pr(X = x_0) + \beta \Pr(Y = y_0) + \gamma \Pr(X = x_0) \Pr(Y = y_0)$. Different correlations between attributes are reflected in the coefficients (e.g. , α, β, γ), but the framework of the method for combining low-dimensional features remains consistent.

This implies that, if we could adaptively adjust the combination coefficients, the method can be transferred to attributes of different distributions and/or correlations. Such adaptiveness can be attained by a pretrained model, which has witnessed enough number of diversified data distributions and could learn to produce the corresponding coefficients with its internal parameters to combine low-level information into high-order joint PDFs. In the literature works for CardEst, this finding has been used to combine the attribute-level information to learn the joint PDF of a single table (IRIS [53]) or a single dataset with updates (ALECE [47]). While these methods have demonstrated feasibility in simpler contexts, they can not be applied to more complex multi-table scenarios. Specifically, IRIS is limited to single-table scenarios with no consideration of the correlations across tables. ALECE is designed as R1D4 a soft lookup table linking SQL queries to the hard-encoded underlying data distribution and cannot adapt to different databases. The details of its limitations are explained in Appendix E of the full version [5].

In this paper, we consider the pretrained CardEst model for the more general multi-table join queries across different databases. It is much more complex than existing works as the model is required to: 1) capture the correlations between attributes in the same single table and across joined tables at the same time; and 2) be adaptive to the difference in the database level, i.e. , different number of tables and attributes, distinct distribution skewness and correlations, and etc. Next, we present the details of the key information to be captured by the pretrained model and the proper tools for building our model.

Key Information Captured by the Pretrained Model. By the above analysis, the CardEst pretrained model learns to represent the high-level joint PDF $\Pr(T_Q)$ and computes the probability $\Pr(Q)$ from $\Pr(T_Q)$ by utilizing a number of features in low-dimensional space w.r.t. each attribute $A_{i,j} \in A_Q$. To this end, the model needs to essentially capture three aspects of key information as follows:

1) *Correlation factors among attributes.* In a database, attributes in the same or different tables are often correlated with others [29, 47, 87], so we could not simply obtain $\Pr(T_Q)$ as $\prod_{A_{i,j} \in A_Q} \Pr(A_{i,j})$. Instead, we must learn some correction factors to elegantly combine $\Pr(A_{i,j})$ on each singleton attribute together to approximate the joint PDF $\Pr(T_Q)$, e.g., using addition or multiplication with learned weights to fit the dependency among attributes.

2) *Scaling factors among tables.* For different tables in a database, the join operation could also change the joint PDF of attributes. As we explained earlier, a tuple in a table T_i may join with a different number of tuples in another table T_j. As a result, for each value v of any attribute $A_{i,j}$ in T_i, the occurring probability of v in T_i may be different from that in $T_i \bowtie T_j$. Therefore, to combine $\Pr(A_{i,j})$ together as $\Pr(T_Q)$, we must also consider the correction effects of the join scaling in our learned weights. This part of information is not explicitly considered in existing works [47, 53].

3) *Filtering factors of predicates.* Upon the joint PDF $\Pr(T_Q)$, we need to extract the probability $\Pr(Q)$ by the predicates in Q. Thus, except simply combining $\Pr(A_{i,j})$ together in terms of the correlation and scaling effects, by Eq. (2), the learned weights in the model should also be able to filter $\Pr(A_{i,j})$ to $\Pr(A_{i,j} \in R_{i,j})$, where $R_{i,j}$ is the constraint region specified by the filter predicate on $A_{i,j}$.

Notably, the three key factors (namely correlation, scaling, and filtering) interact with each other, so the learned model parameters need to condense their information altogether. Meanwhile, we require them to be adaptable to attributes and joins with varying levels of correlation and scaling effects.

Tools to Build the Pretrained Model. Although the above task seems to be very difficult, we borrow the successful experience of applying the *self-attention mechanism* [72] in the pretrained NLP models [16, 20, 50] to fulfill such goals. The self-attention mechanism allows models to learn to concentrate on relevant aspects of input data while disregarding the irrelevant, thereby mimicking human cognitive attention processes.

R3D4 In particular, let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times d}$ be input data with n rows, where each row $\boldsymbol{x}_i$ represents an input token (e.g., a word embedding) with the same dimension d. The self-attention mechanism transforms this input matrix into three matrices: Key ($\boldsymbol{K}$), Query ($\boldsymbol{Q}$), and Value ($\boldsymbol{V}$) through learned weight matrices: $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$, $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$ and $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$, where $\boldsymbol{W}^K \in \mathbb{R}^{d \times d_k}$, $\boldsymbol{W}^Q \in \mathbb{R}^{d \times d_q}$, and $\boldsymbol{W}^V \in \mathbb{R}^{d \times d_v}$ are learnable matrices for keys, queries, and values, respectively, and $d_q = d_k$ usually. Denote $\boldsymbol{k}_i$, $\boldsymbol{q}_i$, and $\boldsymbol{v}_i$ as the i-th row of the matrices $\boldsymbol{K}$, $\boldsymbol{Q}$, and $\boldsymbol{V}$, respectively. The attention coefficients $\alpha_{i,j}$, which reflect the importance from $\boldsymbol{x}_j$ to $\boldsymbol{x}_i$, are computed as $\alpha_{i,j} = \exp(e_{i,j}/\sqrt{d_k}) / \sum_{l=1}^{n} \exp(e_{i,l}/\sqrt{d_k})$, where $e_{i,j} = \boldsymbol{q}_i \boldsymbol{k}_j^\top$ is the correlation coefficient between query i and key j and $\sqrt{d_k}$ is a normalization factor. Based on these scores, we can obtain a number of output tokens $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n$, where $\boldsymbol{y}_i = \sum_j \alpha_{i,j} \boldsymbol{v}_j$, to condense low-level information in $\boldsymbol{X}$ to high-order representations.

In practice, the *self-attention mechanism* can be implemented multiple times to create a *multi-head self-attention mechanism.* In the NLP domain, it is applied to compress the semantics of word embeddings to represent phrases, sentences and documents. For our CardEst task, we apply it to condense the information of attributes, join relations and filtering predicates together to represent the joint PDF $\Pr(T_Q)$ in high-dimensional space and compute the query probability $\Pr(Q)$. Notably, it fully attains the requirements to capture the key information in the pretrained model.

On one side, for different input embedding vectors, the attention coefficients $\alpha_{i,j}$ for combination are different. Therefore, the model could produce adaptive weights to: 1) compress attributes and joins with different correlations and distributions; and 2) focus on different parts of the joint PDF representation w.r.t. different filtering conditions. This ensures the pretrained model is applicable to different databases.

On the other side, the size of the learned matrices $\boldsymbol{W}^K$, $\boldsymbol{W}^Q$ and $\boldsymbol{W}^V$ are only determined by the embedding dimensions d, d_k, d_q, d_v, but has no relation with the input length n. Therefore, it naturally hinders the difference between databases and queries with varied numbers of tables, attributes and filtering predicates.

4 OUR PRETRAINED MODEL

M5

We outline our proposed Pretrained Multi-Table CardEst Model (PRICE) in Figure 1. On a high level, for an input query Q, PRICE applies a number of features in low-dimensional space (see details below). These attributes could be easily obtained at a low cost and widely adapted to any database. Then, the model maps such input features into embedding vectors with a fixed length by simulating the CardEst process in two stages. The *joining stage* simulates the joining process across multiple tables in Q. It applies a self-attention module to integrate the distribution and scaling information over all attributes in the join conditions. The output reflects the backbone information of the joined table T_Q (or the joint PDF $\Pr(T_Q)$). Then, the *filtering stage* utilizes another self-attention module to fuse the backbone information with the distribution and filtering predicate information over all other attributes together. The outputs naturally condense the information of correlation, scaling, and filtering predicate over all attributes together and thus offer a rich representation w.r.t. $\Pr(T_Q)$ and $\Pr(Q)$. Based on such output and some auxiliary query-level features, we could finally obtain the desired cardinality $\mathrm{Card}(Q)$. The details on feature selection and model workflow are described as follows.

M4 R3W1 R3D2

Feature Selection. For a database D and a SQL query Q, our PRICE takes four classes of features to capture key information as follows:

1) *The value distribution* $\Pr(A_{i,j})$ *of each attribute* $A_{i,j} \in A_Q$. For continuous attributes, we use a histogram vector to represent its distribution. To eliminate the different ranges of attributes, we normalize each histogram vector to span into $[0, 1]$. For categorical attributes, we utilize the SpaceSaving Summary [55] to maintain its distribution. It records the frequency of each item and can be accessed and updated at the same cost as the histogram vector. These vectors of distributions are set to the same length. They are regarded as the basic *tokens* in our approach, similar to the basic word embeddings in the NLP models.

M4 R3W1 R3D2

2) *The distribution of the scaling factors of each join condition* $T_i.A_{i,x} = T_j.A_{j,y}$ *in* Q. Let s be a tuple in table T_i. The *scaling factor* of s refers to how many of tuples t in table T_j could join with s, i.e. $s[A_{i,x}] = t[A_{j,y}]$. We also encode the distributions of scaling factors as histogram vectors. They capture the inter-table correlation information and help the model learn how to scale $\Pr(T_i)$ to $\Pr(T_Q)$. Details on the calculation of scaling factors and the associated maintenance cost analysis are provided in Appendix A [5].

M4 R3W1 R3D2 R1W2 R3D1

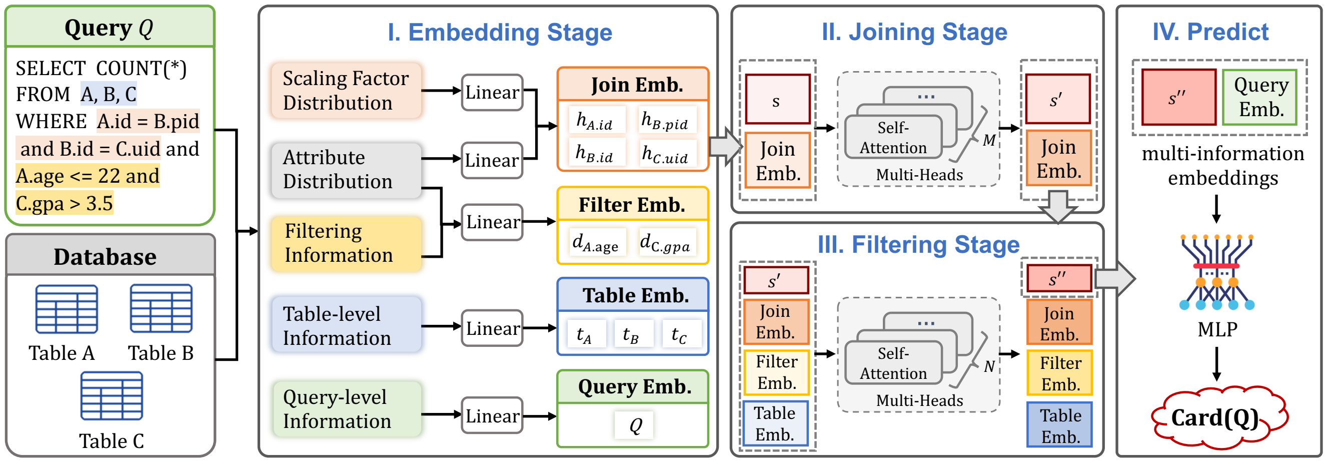


Figure 1: Model architecture of PRICE.

3) *The information of each filtering predicate* "$T_i.A_{i,j}$ OP VALUE" *in* Q. We encode each filtering predicate as a three-dimensional vector comprising: i) Range: represented as a two-dimensional vector [lower_bound, upper_bound], normalized to the attribute's domain. For example, "A.age<22" with domain [0, 100] is encoded as [0, 0.22]. For categorical attributes, the range is the index of the bin containing the VALUE. If the predicate is an equality, the lower and upper bounds are set to the same value; and ii) Selectivity: An estimated decimal value $\in$ [0, 1], calculated as the ratio of tuples satisfying the predicate on attribute $A_{i,j}$ over all tuples in T_i. This aids in computing $\Pr(Q)$ from the joint PDF.

M4
R3W1
R3D2

4) *The table-level and query-level auxiliary information of* Q. For each table T_i in Q, we encode the simplified single-table selectivity estimates produced by heuristic estimators, including AVI, MinSel, and EBO (see details in [23]). We also encode the size $|T_i|$ of each table to characterize its volume. For query Q, we encode: i) the number of tables and joins in Q; and ii) the cardinality estimated by traditional methods, e.g., the histogram-based methods in PostgreSQL. In this way, we feed the wisdom of the traditional CardEst methods into the model, which could provide more guidelines to produce better results.

Notably, all of the above features could be easily obtained in $O(\sum_i |T_i|)$ time. Thus, our PRICE is as cheap as the efficient traditional 1-D histogram-based methods in terms of feature construction. Meanwhile, when data changes, these features could be incrementally updated in almost real-time.

Workflow of PRICE. As shown in Figure 1, PRICE consists of three main stages: the embedding stage for data preparation and the joining and filtering stages for data integration.

1) *Embedding Stage:* Initially, PRICE maps the input features for each attribute and table into fixed-dimensional embeddings using simple linear models. Specifically, for each attribute $A_{i,j}$ that occurs in any join condition of Q, its value distribution vector and scaling factor distribution vector are separately mapped into a fixed-dimensional vector by different linear models. These vectors are then concatenated to form the embedding vector $\boldsymbol{h}_{i,j}$. For each attribute $A_{i,j}$ that occurs in any filtering predicate of Q, we concatenate its value distribution vector, filter range, and selectivity, then map this combined vector to a fixed dimension embedding vector $\boldsymbol{d}_{i,j}$ using a linear model. For each table T_i, we apply a linear model with shared parameters to map its table-level auxiliary information into an embedding $\boldsymbol{t}_i$. For the query Q, we also apply a linear model with shared parameters to map its query-level auxiliary information into an embedding $\boldsymbol{Q}$. Additionally, we randomly initialize a special embedding vector $\boldsymbol{s}$ as input. $\boldsymbol{s}$ would be fed into the subsequent modules with the other embedding vectors together. It serves to amalgamate the information of other embedding vectors together, akin to the CLS token in NLP tasks [20] or the virtual node in Graph Neural Networks (GNN) [83]. We will explain the detailed role of $\boldsymbol{s}$ later. Notice that the dimension of all embedding vectors $\boldsymbol{h}_{i,j}$, $\boldsymbol{d}_{i,j}$, $\boldsymbol{s}$ and $\boldsymbol{t}_i$ are uniformly fixed, ensuring the generality across different numbers of joins and filtering predicates in the query. Then, PRICE applies two main stages to exploit the capabilities of the multi-head self-attention mechanism to capture key information for CardEst.

M4
R3W1
R3D2

2) *Joining Stage (Backbone Assembly):* We integrate the information related to all attributes in the join conditions to build the backbone structure of the table T_Q. The token $\boldsymbol{s}$ and all embedding vectors $\boldsymbol{h}_{i,j}$—representing each attribute $A_{i,j}$ in the join condition—are fed into a multi-head self-attention module. The module converts them into new embedding vectors $\boldsymbol{s}'$ and $\boldsymbol{h}'_{i,j}$. Each $\boldsymbol{h}'_{i,j}$ (as well as $\boldsymbol{s}'$) is a weighted sum over all embedding vectors $\boldsymbol{h}_{i,j}$ and $\boldsymbol{s}$. The weights, a.k.a. attention coefficients, are adaptively determined by learned parameters (see details in Section 3). Using this attention mechanism, the output vectors $\boldsymbol{h}'_{i,j}$ and $\boldsymbol{s}'$ capture two sides of information together: i) the correlations between the scaling factor distributions of different joins in Q; and ii) the impact of the joining operations (i.e., the scaling effects) on the value distribution of attributes. This combined knowledge is represented as the outputs $\boldsymbol{h}'_{i,j}$. Upon them, the model could then know how to scale multiple low-level PDFs $\Pr(T_i)$ on each single table to the high-level joint PDF $\Pr(T_Q)$ on the joined table in the subsequent steps.

M4
R3W1
R3D2

3) *Filtering Stage (Information Refinement):* The focus shifts to refining the backbone information with all filtering conditions to obtain accurate estimation results. Specifically, we aim to obtain the information of $\Pr(T_Q)$ and compute the probability $\Pr(Q)$ from $\Pr(T_Q)$. Recall that each embedding vector $\boldsymbol{d}_{i,j}$ condenses the value

distribution and filtering information on each attribute $A_{i,j}$ occurring in the filtering predicates, and each embedding vector $\boldsymbol{h'}_{i,j}$ condenses the information of the scaling factor distributions of all joins. In this stage, we apply another multi-head self-attention module to integrate their information together. In particular, let $U = \{\boldsymbol{d}_{i,j}\} \cup \{\boldsymbol{h}'_{i,j}\} \cup \{\boldsymbol{s}'\} \cup \{\boldsymbol{t}_i\}$ be the set of all input embedding vectors. Then, for each $\boldsymbol{u} \in U$, the self-attention module obtains an embedding vector $\boldsymbol{u}'$ using a weighted sum over all vectors in U.

M4
R3W1
R3D2

We find that, the vectors $\boldsymbol{u}'$ integrate three sides of information together: i) the scaling information of multiple tables (encoded in $\boldsymbol{h}'_{i,j}$) are fused with the value distributions of other attributes (encoded in $\boldsymbol{d}_{i,j}$) not occurring in the join conditions. This comprehensively amalgamates the information to represent the joint PDF $\Pr(T_Q)$; ii) the information of all filtering predicates (also encoded in $\boldsymbol{d}_{i,j}$), which reflects how to compute $\Pr(Q)$ from $\Pr(T_Q)$; and iii) some coarsen-grained information for estimated cardinality on each single table (encoded in $\boldsymbol{t}_i$). This would more or less provide the representation vectors with some guidelines from the traditional CardEst methods to correct $\Pr(Q)$. In short, the output vectors $\boldsymbol{u}'$ encapsulate all relevant information to offer a rich representation w.r.t. $\Pr(T_Q)$ and $\Pr(Q)$.

Based on these, we are prepared to obtain the final estimated cardinality. Notice that, it is not necessary to utilize all output vectors $\boldsymbol{u}'$ produced by the filtering stage. Instead, let $\boldsymbol{s}''$ be the corresponding output vector to $\boldsymbol{s}'$. We only need to apply $\boldsymbol{s}''$ into the final step as $\boldsymbol{s}''$ already condenses the information of other input vectors through the self-attention mechanism. As a result, $\boldsymbol{s}''$ and the query-level embedding Q (serving as guidelines) are concatenated and then fed into a non-linear neural network, specifically referring to a multilayer perceptron (MLP) [31]. This concatenated input is used to obtain the desired cardinality $\text{Card}(Q)$.

M4
R3W1
R3D2

Analysis Results of PRICE. To better understand the functionality of the attention mechanism in PRICE, we present some analysis results with concrete examples. The representative examples in Figure 2 and Figure 3 are extracted from our test dataset STATS, and additional ones are given in Appendix B [5]. The darker colors indicate larger attention weights or other values.

M1
R1W1
R2W1
R2D2

At first, we find that, the attention weights contain valuable information for accurate CardEst. Based on our observations, *queries with diversified attention weights tend to result in more accurate estimates.* For example, the query in Figure 2(a) with varied attention weights over features has a much lower estimation error than the query in Figure 2(b) with uniform attention weights. Their q-errors (defined in Section 6.1) are 1.97 and 2079, respectively. This indicates that the learned attention weights help to differentiate the influence of various features and extract useful information to estimate the final cardinality. Meanwhile, this observation also provides practical guidance for deployment: if the model fails to assign adaptive weights to a large number of queries in a new dataset, fine-tuning may be necessary; otherwise, traditional CardEst methods embedded in the DBMS may be more reliable.

Then, with an in-depth analysis, we exhibit how these attention weights play their roles in CardEst. We find that *PRICE could automatically and adaptively pay attention to important features and adjust its coefficients to fit different input distributions.*

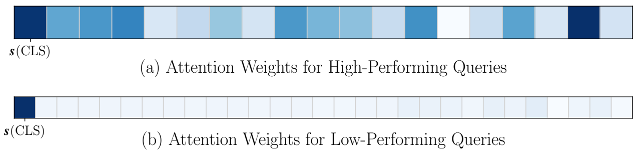
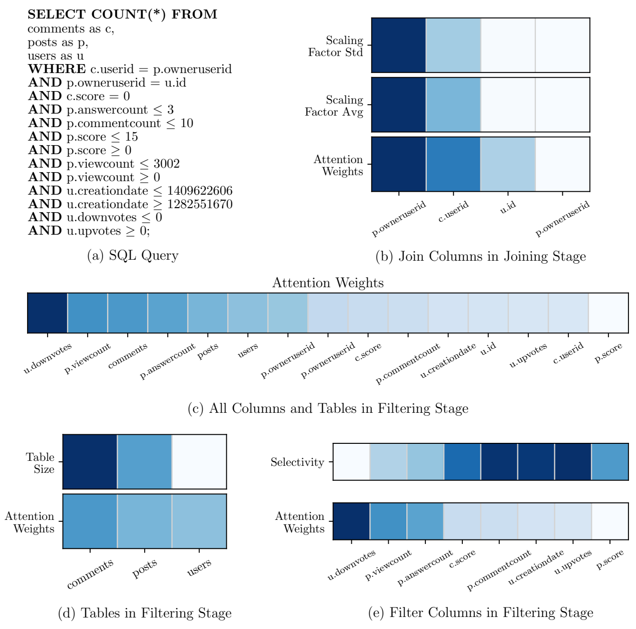


Figure 2: Attention weights impact on model performance.



```
SELECT COUNT(*) FROM
comments as c,
posts as p,
users as u
WHERE c.userid = p.owneruserid
AND p.owneruserid = u.id
AND c.score = 0
AND p.answercount ≤ 3
AND p.commentcount ≤ 10
AND p.score ≤ 15
AND p.score ≥ 0
AND p.viewcount ≤ 3002
AND p.viewcount ≥ 0
AND u.creationdate ≤ 1409622606
AND u.creationdate ≥ 1282551670
AND u.downvotes ≤ 0
AND u.upvotes ≥ 0;
```

Figure 3: Attention weight for a specific query.

On the one hand, our analysis reveals that *PRICE's attention mechanism could adaptively assign weights to different tables and attributes based on their significance to the final cardinality.* A higher attention weight on a token t indicates that PRICE pays more attention to t to obtain accurate estimation results. For the example query in Figure 3(a), in the first joining stage, PRICE assigns different attention weights to attributes involved in the join conditions to learn the backbone information of the whole joined table. Figure 3 (b) illustrates the relations between the attention weights, the mean and the standard derivation of the scaling factor distributions of these join attributes. Obviously, the attention weights are positively correlated with the mean value and standard derivation of the scaling factor distributions of these join attributes. We find that: *i) join attributes with larger mean scaling factors or more diversified scaling factors (i.e. , larger standard deviations) receive higher attention weights.* This is because these attributes contain tuples having larger scaling factors, occur more frequently in the joined table and thus significantly impact the final cardinality.

In the second filtering stage, PRICE refines the backbone information with the value distributions and filtering information of all other attributes and tables to obtain accurate estimation results. In Figure 3 (c), we rank the attention weights over all tables and attributes in this stage. We observe that: *ii) The model primarily focuses on attributes having filtering conditions (e.g. , u.downvotes and p.viewcount),* as the model requires fine-grained filtering over these attributes to obtain accurate cardinality.

Table 1: **Overview of test datasets: database statistics and workload specifications.**

Dataset Name	Database Statistics										Workload Specifications			
	Sectors	# of Tables	# of Cols (All Tables)	# of Rows (All Tables)	# of Join Relations	Volume	Total Attribute Domain Size	Distribution Skewness (MIN/AVG/MAX)	Average Pairwise Correlation	Join Forms	Joined Tables	# of Filtering Predicates	Join Type	True Cardinality Range
IMDB	Entertainment	6	37	$6.2 \cdot 10^7$	15	3.0G	$8.5 \cdot 10^7$	-1.9/19.2/384.5	0.32	Star	2 - 5	0 - 4	PK-FK/FK-FK	$2 - 1{\cdot}10^{10}$
STATS	Education	8	43	$1.0 \cdot 10^6$	30	32.3M	$1.9 \cdot 10^6$	-1.0/10.5/134.5	0.46	Mixed	2 - 7	0 - 10	PK-FK/FK-FK	$16 - 8{\cdot}10^{10}$
ErgastF1	Sport	14	98	$5.5 \cdot 10^5$	67	15.1M	$3.2 \cdot 10^5$	-2.0/2.2/66.6	0.44	Mixed	2 - 7	0 - 9	PK-FK/FK-FK	$1 - 2{\cdot}10^{10}$
VisiualGenome	Education	6	20	$3.6 \cdot 10^6$	6	73.4M	$3.1 \cdot 10^5$	-3.8/-0.4/2.0	0.20	Mixed	2 - 6	0 - 7	PK-FK/FK-FK	$24 - 2{\cdot}10^8$

In detail, Figure 3 (d) and (e) show the relations between attention weights with table size and selectivity of attributes in the filtering conditions, respectively. We further find that: *iii) PRICE tends to assign higher attention to larger tables* since they typically have a more significant impact on the cardinality of the query results; and *iv) The model focuses more on filtering attributes with lower selectivity* since decreasing the selectivity would increase the difficulty of accurate estimation (in similar to sampling-based methods). These four findings reveal that PRICE could learn to automatically prioritize elements by their impacts to the final cardinality.

On the other hand, we find that, *in the process of fusing multiple tables together, PRICE could appropriately adjust its attention weights based on the joined table's distribution.* For instance, in the query shown in Figure 3, the filtering condition "u.downvotes ≤ 0" shows high selectivity in the single table "user" (98%). According to the above third finding, PRICE tends to give lower weight to the column with high selectivity. However, in this case, it also assigns a large weight to this attribute. This is because this filtering condition incurs very low selectivity in the joined table (1.5%), which requires the model to pay more attention to obtain accurate estimation results. This suggests that PRICE would appropriately adjust its attention weights on attributes when a distribution drift exists between the single and joined tables.

5 DATA COLLECTION AND MODEL TRAINING

In this section, we introduce how to collect different databases and the related query workloads to train our model PRICE. We have made both the collected databases and query workloads (including queries and their corresponding cardinalities) as well as the pretrained model publicly available in the repository [12] to nourish the community. We hope the collected databases and query workloads could serve as a new and comprehensive benchmark for evaluating CardEst methods.

M3
M5
R2D1

Datasets Collection. To effectively assess the performance of our PRICE, it is imperative to train and test the model across various datasets. However, existing CardEst methods commonly utilize a limited number of datasets for evaluation, which are insufficient for the expansive training needs of the pretrained model. To cultivate a model capable of generalizing across diversified databases, we exhaustively searched the public data repositories and obtained 30 datasets, including: 1) 6 well-established benchmark datasets for evaluating CardEst methods, such as TPC-H [10], SSB [61], IMDB [6] and STATS [13]; and 2) 24 new datasets assembled from [58]. We clean and process each dataset. We observe that these datasets closely resemble real-world scenarios and demonstrate notable diversity. Table 1 provides detailed properties of the four test datasets and their corresponding workloads. Due to space limits, we put the comprehensive details of all datasets are provided in Appendix C [5].

M3
M5
R2D3

Query Workloads Generation. For model training and evaluation, corresponding workloads for each dataset must be prepared. We retain the testing workloads JOB-light[7] and STATS-CEB[29] for widely recognized datasets IMDB and STATS, respectively, and developed new training and testing workloads for others by the following process. Specifically, for each dataset, we obtain a graph of its join schema where each node represents a table T_i and each edge connecting T_i and T_j represents a join relation between two tables. We enumerate all connected subgraphs from the join schema graph, where each subgraph represents a possible join relation among some tables in the dataset. To generate an SQL query, we randomly select a subgraph and then attach it with some filtering predicates. In particular, we first collect all categorical and numerical attributes in all the tables that occurred in the subgraph. Let m be the total number of attributes. We uniformly sample a number n from 1 to m and then repeatedly sample n attributes at random. For each sampled attribute A_i, if it is numerical, we pick its lower bound l and upper bound u uniformly at random and set the filtering predicate as $A_i \leq u$ and $A_i \geq l$; if it is categorical, we randomly select a value v from its domain and set the filtering predicate as $A_i = v$. For each dataset, we generate 5×10^4 SQL queries for model training. We obtain the true cardinality of each query through actual execution. The detailed Pseudo-code for workload generation is given in Appendix D [5].

M3
M5
R2W3
R2D1

Model Pretraining. We reserve the four widely applied datasets—IMDB, STATS, ErgastF1 and VisualGenome—as the test datasets and apply the remaining 26 datasets with the generated query workloads to pretrain our model PRICE. The details are as follows:

1) *Loss Function*: Due to the board range of the cardinality (e.g., from 1 to 10^{11}), we apply a logarithmic transformation to normalize its scale and use $\log(\mathrm{Card}(Q))$ as the target label for training. The Mean Squared Error (MSE) function is employed as the loss function during training. That is, for a batch of k SQL queries $Q_1, Q_2, \ldots, Q_k$, the total loss is $(\sum_{k'=1}^{k} (\log(\mathrm{Card}(Q_{k'})) - \log(\widehat{\mathrm{Card}}(Q_{k'})))^2)/k$.

2) *Model Hyperparameters*: We implement our model and its training function in Python 3.10. In our PRICE, we use a 40-dimensional vector to represent all input features on value distribution and scaling factor distribution of attributes. The dimension of all inner embedding vectors $\boldsymbol{h}_{i,j}$, $\boldsymbol{h'}_{i,j}$, $\boldsymbol{d}_{i,j}$, $\boldsymbol{s}$, $\boldsymbol{s'}$, $\boldsymbol{s''}$ and $\boldsymbol{t}_i$ (see details in Section 4) are set to 256. The self-attention modules applied in the joining and filtering stage all comprise 8 heads.

3) *Training Method*: To promote stable learning and robust generalization across databases, we shuffle the training data and use a large batch size (i.e., 1.5×10^4). We adopt the Adam optimizer [40] for model training with the initial learning rate of 2.85×10^{-5} and a weight decay of 5×10^{-5}. To stabilize the training process, we utilize the StepLR function in PyTorch to adaptively adjust the learning rate. To prevent overfitting, we also apply the dropout function.

Table 2: Overall performance of different CardEst methods.

DATASETS	CARDEST METHOD	E2E TIME (S)	Q-ERROR 50%	80%	90%	95%	99%	P-ERROR 50%	80%	90%	95%	99%	MODEL SIZE (MB)	(PRE)TRAINING TIME (MIN)	INFERENCE TIME (MS)
IMDB	PG	4037	1.95	6.04	19.04	49.44	879.64	**1.00**	1.31	1.73	2.45	13.44	–	–	–
	ALECE	3911	1.75	4.21	11.49	19.07	124.68	**1.00**	1.07	1.46	1.92	5.44	233.11	472.88	5.08
	MSCN	4755	3.24	12.75	28.54	104.20	411.90	1.07	1.71	2.31	4.00	15.66	**1.57**	75.19	**0.79**
	DeepDB	3850	1.31	2.97	3.61	**5.03**	**14.03**	**1.00**	**1.00**	1.09	1.27	1.56	88.33	73.73	4.73
	NeuroCard	3664	1.66	4.14	7.80	14.25	22.22	**1.00**	1.23	1.78	2.32	4.05	50.27	14.55	18.43
	FactorJoin	3588	13.86	89.88	348.96	1027.19	4794.80	1.04	1.61	2.86	3.46	7.91	5.59	78.45	3.20
	PRICE (Pretrained)	3910	1.77	4.07	8.39	15.45	70.88	**1.00**	1.05	1.16	1.28	1.63	41.14	313.56	5.27
	PRICE (Finetuned)	**3454**	**1.29**	**2.05**	**2.92**	5.36	29.45	**1.00**	**1.00**	**1.04**	**1.08**	**1.16**	41.14	**6.47**	5.27
	Optimal	3442	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	–	–	–
STATS	PG	30484	1.87	7.11	20.71	73.35	1600.22	1.04	1.67	2.44	4.16	20.57	–	–	–
	ALECE	25716	1.67	3.79	7.93	16.44	118.96	**1.00**	1.38	1.84	2.17	3.49	233.04	93.09	10.88
	MSCN	42051	6.19	70.02	363.71	1609.45	$9.15{\cdot}10^4$	1.36	2.64	5.98	13.41	59.14	**1.60**	2.03	**0.81**
	DeepDB	22931	1.84	19.94	73.52	132.63	1507.78	1.03	2.17	2.85	4.41	9.32	249.93	142.12	12.42
	NeuroCard	24435	2.12	12.57	48.40	228.41	$1.32{\cdot}10^4$	1.03	1.56	1.97	2.92	8.22	121.88	36.43	33.80
	FactorJoin	50179	5.41	40.44	150.52	666.59	$2.10{\cdot}10^4$	1.09	2.05	3.07	5.19	21.73	1.86	**0.55**	23.70
	PRICE (Pretrained)	18292	1.87	5.49	12.46	35.55	579.67	**1.00**	1.42	1.90	2.41	6.40	41.14	313.56	14.13
	PRICE (Finetuned)	**17192**	**1.41**	**2.28**	**3.67**	**7.07**	**42.03**	**1.00**	**1.00**	**1.25**	**1.46**	**2.33**	41.14	10.20	14.13
	Optimal	17160	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	–	–	–
ErgastF1	PG	48185	1.60	4.26	11.30	27.47	114.24	**1.00**	1.32	1.64	1.98	6.39	–	–	–
	ALECE	50902	1.75	3.15	5.18	8.43	53.81	1.03	1.32	1.61	2.08	3.35	233.34	121.49	7.06
	MSCN	45511	10.20	99.89	353.02	999.52	$1.32{\cdot}10^4$	1.52	2.54	4.92	8.61	13.72	**1.66**	**5.08**	**0.81**
	PRICE (Pretrained)	44532	**1.43**	3.12	6.45	14.57	67.97	**1.00**	1.15	1.46	1.73	**2.24**	41.14	313.56	7.14
	PRICE (Finetuned)	**44350**	**1.43**	**2.46**	**4.06**	**7.37**	**29.96**	**1.00**	**1.00**	**1.42**	**1.57**	2.28	41.14	5.21	7.14
	Optimal	44256	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	–	–	–
Visual Genome	PG	6165	1.17	4.34	15.12	383.09	$1.17{\cdot}10^4$	1.04	1.04	1.44	2.26	4.72	–	–	–
	ALECE	5874	1.13	1.34	1.44	1.58	2.01	**1.00**	**1.00**	1.04	1.04	**1.08**	232.72	49.68	6.25
	MSCN	12070	2.84	9.34	18.91	47.45	108.48	2.68	2.74	4.38	4.39	4.40	**1.54**	39.86	**0.78**
	NeuroCard	6673	1.04	12.23	14.35	26.75	53.75	1.00	1.20	1.21	1.28	2.73	38.38	12.91	9.85
	PRICE (Pretrained)	5840	1.65	3.59	5.17	15.67	115.61	**1.00**	**1.00**	1.01	1.42	2.64	41.14	313.56	6.22
	PRICE (Finetuned)	**5838**	**1.06**	**1.19**	**1.29**	**1.35**	**1.54**	**1.00**	**1.00**	**1.00**	**1.00**	**1.08**	41.14	**11.48**	6.22
	Optimal	5834	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	–	–	–

4) *Training Environment*: We train our model on a Linux server outfitted with a 96-core Xeon(R) Platinum 8163 CPU @ 2.50GHz, 512GB of main memory, and 8 NVIDIA A100 GPUs. For the whole training workload having $1.3{\times}10^6$ SQL queries, we consume around 5 hours to produce a model of around 40MB.

6 EXPERIMENTAL STUDIES

This section presents a comprehensive evaluation of PRICE against existing state-of-the-art (SOTA) techniques for multi-table cardinality estimation. The evaluation results aim to answer the following pivotal questions about its performance:

1) How does our pretrained model PRICE perform when directly applied to unseen databases with little preparation? (Section 6.2)

2) How does PRICE perform after finetuning on specific databases compared to existing ML-based CardEst methods? (Section 6.3)

3) Can PRICE generalize to data updates and query workload drifts? (Section 6.4)

4) What are the impacts of training data on PRICE? (Section 6.5)

M2 R1W3 R3D5 5) How do individual features contribute to PRICE's performance? (Section 6.6)

6.1 Experiment Setup

Datasets. We selected four real-world datasets—IMDB, STATS, ErgastF1 and VisualGenome—for evaluation. They are widely adopted to examine the performance of CardEst methods in the literature [29, 35, 60]. The detailed properties of these datasets are presented in Table 4. STATS [13, 29] is an anonymized compilation of all user-contributed content from the Stats Stack Exchange network [11]. The associated query workload STATS-CEB contains 146 queries for testing. The IMDB dataset [44] is on movies and stars with the simplified JOB-light workload [7] containing 70 realistic queries. ErgastF1 [3] provides extensive information on Formula 1 races spanning from the 1950 season to the present day. VisualGenome [14] encompasses dense annotations of objects, attributes, and relationships within various genes. We use the method in Section 5 to generate 148 and 186 queries on ErgastF1 and VisualGenome for testing, respectively.

Competing Methods. We select the competing CardEst methods based on their performance reported in various benchmarks [29, 70] and evaluation studies [78, 81, 87]. For each category, namely traditional, data-driven, query-driven and hybrid CardEst methods, we select the representative ones as follows:

1) **PG** stands for the basic 1-D histogram based approach used in PostgreSQL [26] for CardEst. This simple method serves as a baseline to measure the effectiveness of more sophisticated methods.

2) **NeuroCard** [81] builds a single deep auto-regression model on the joint PDF of all tables in the database. The cardinality of any query could then be accessed from this model.

3) **DeepDB** [36] learns the joint PDF of data using Sum-Product Networks (SPNs) [63]. It builds several SPNs, each covering a subset of tables for CardEst.

4) **FactorJoin** [77] applies a cohesive factor graph to combine the learned cardinality on every single table, and the histograms of joins together for CardEst on multiple tables.

5) **MSCN** [41] utilizes a multi-set convolutional network to amalgamate data and query information together to learn cardinality.

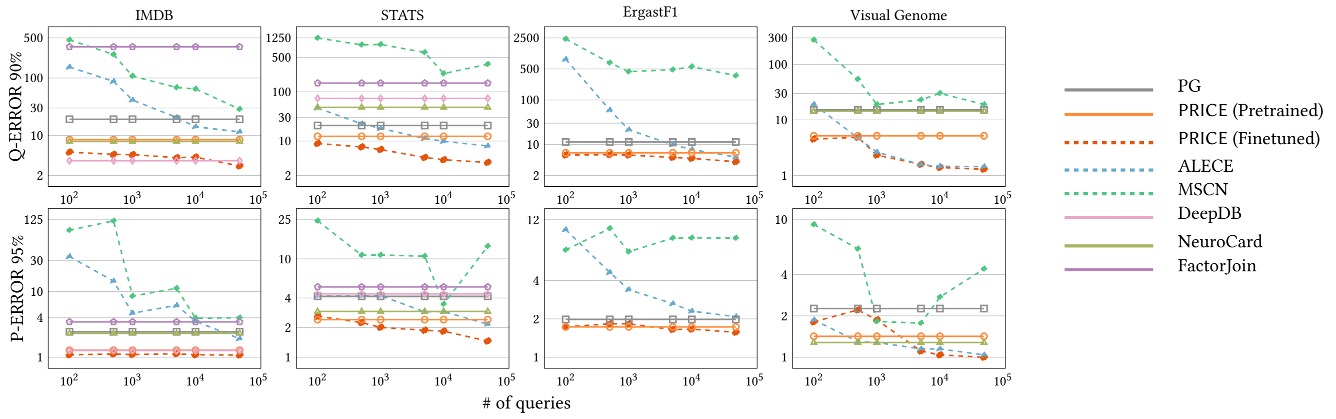


Figure 4: Estimation errors of cardinality estimation models across varying numbers of training queries.

R1D4

6) **ALECE** [47] is designed as a soft lookup table, linking each SQL query to the underlying data distribution to estimate the cardinality of the query using attention mechanisms.

For these methods, we apply their open-source implementations [1, 2, 4, 8, 9] and follow their original configurations for parameter setting. For DeepDB, we set its RDC independence threshold to 0.3 and split each SPN node with at least 1% of the input data; for NeuroCard, the sampling size is set to 8,000. We do not execute DeepDB and FactorJoin on the ErgastF1 and VisualGenome datasets as DeepDB only supports PK-FK joins, and the implementation of Factorjoin is hard-coded for the IMDB and STATS datasets.

Besides them, an **Optimal** approach, which utilizes the true cardinality for query plan generation, is also included for benchmarking. As shown in [44], this method could find the optimal plans for most of the queries. Therefore, it could serve as a borderline to exhibit the best performance that the CardEst could bring for query optimization. We inject all CardEst methods into PostgreSQL 13.1 using PilotScope [86]—a middleware for deploying ML algorithms into databases—for end-to-end testing in actual DBMS.

Evaluation Metrics. We evaluate the effectiveness of each CardEst method by the following metrics:

1) **End-to-end (E2E) Time** refers to the total time cost of plan generation using estimated cardinalities and physical plan execution. It reflects the performance gain that a CardEst method could bring to the query optimization process. Therefore, it is regarded as the gold standard metric to gauge the effectiveness of CardEst methods [29].

2) **Q-ERROR** [57] assesses the relative multiplicative deviation of the estimation results from the exact value as Q-ERROR = $\max(\widehat{\text{Card}}(Q)/\text{Card}(Q), \text{Card}(Q)/\widehat{\text{Card}}(Q))$. It could reflect the accuracy of estimation results in a coarse-grained manner. However, it can not distinguish the cases of under-estimate and over-estimate and the impacts of estimation errors of different sub-queries w.r.t. the plan generation.

3) **P-ERROR** [29] corrects the drawbacks of Q-ERROR and could be computed without the actual execution of queries. By [44], the plan cost estimated by the cost model is a good surrogate for the execution time. Therefore, we evaluate the quality of a CardEst method by comparing the execution cost of the plans derived from the estimated and true cardinalities. Specifically, for a query Q, let C^E and C^T denote the estimated and true cardinalities of all sub-queries of Q. Let P^E and P^T be the plan generated by the query optimizer by feeding C^E and C^T into it. Let M be the cost model of the query optimizer and $M(P, C)$ denote the estimated cost of plan P with cardinality C. $M(P, C)$ often approximates the execution time of plan P. Obviously, $M(\text{P}^T, \text{C}^T)$ is the estimated cost of executing the (most possible) optimal plan P^T. In the actual execution of plan P^E, the exact cardinality C^T would be instantiated, so the estimated execution cost of plan P^E is $M(\text{P}^E, \text{C}^T)$. Thus, we define it as P-ERROR $= M(\text{P}^E, \text{C}^T)/M(\text{P}^T, \text{C}^T)$. As analyzed in [29], P-ERROR considers the impacts of estimation errors of all sub-queries in terms of plan optimization and is more highly correlated to the E2E time than Q-ERROR.

Besides them, we also consider the **Training Time** for model training, the **Inference Time** for accessing cardinality from the CardEst models, and the **Model Size** for the amount of memory used to store model parameters. These metrics could reflect the performance of each CardEst method from multiple views.

6.2 Performance on Unseen Datasets

We compare our PRICE and other CardEst methods on the four unseen new databases. For fairness, we also apply 5×10^4 queries on each dataset to train MSCN and ALECE. As shown in Table 2, we have the following observations:

R3W2

1) Our pretrained model, PRICE, demonstrates competitive performance compared to other CardEst methods across various metrics. Specifically, for the most crucial E2E time, PRICE surpasses the basic PG and other ML-based methods in almost all cases (except IMDB, where all methods attain good results). On STATS, the E2E time of PRICE is 1.67×, 1.25×, and 1.41× faster than PG, DeepDB, and ALECE, respectively. In terms of the Q-ERROR and P-ERROR metrics, PRICE is always better than PG and has no significant gaps with other ML-based methods. This verifies the success of the design choices of our PRICE model and its pretraining strategies.

2) Our PRICE consistently demonstrates strong and stable performance across all datasets, while existing ML-based methods may exhibit significant variability. For instance, FactorJoin achieves near-optimal performance on IMDB and performs even worse than PG on STATS. This property of PRICE plays an important role when actually deployed into DBMS. By [74], the DBMS not only focuses

on performance gains but also pays attention to avoiding performance regressions. The reason for that is our model encounters a number of different data distributions and query workloads during pretraining, so the learned parameters could generalize to attain stable performance on any database. Whereas, existing ML-based methods often have prior assumptions on the data distributions (e.g., DeepDB assumes attributes are not highly correlated), so they may fail when the dataset does not obey such assumptions.

3) PRICE is time and space efficient. Its inference time spans from several to a dozen milliseconds, which would not degrade the plan generation process. In the actual deployment, we only need to store one singleton PRICE model having 41.14MB for all databases. This is much smaller than storing the models trained by ALECE, NeuroCard, and DeepDB on each specific database. The pretraining cost of PRICE is longer than other ML-based methods. However, we just need to train it once, and then the model can be seamlessly applied to any database.

In short, this set of experiments verifies the superiority of our PRICE over other CardEst methods. It generates plans with much higher and more stable quality, incurs less time and space overheads, and exhibits the easiness for system deployment. To further demonstrate PRICE's effectiveness, we evaluated it on more challenging workloads (JOB-Adapt), where it continued to perform well; detailed results are provided in Appendix F [5].

M2

R2W2

6.3 Performance of PRICE with Finetuning

We further examine the performance of our PRICE after finetuning. Figure 4 illustrates the estimation errors (the 90%-quantile Q-ERROR and 95%-quantile P-ERROR) of different CardEst models by varying the number of training queries. We omit the results of other metrics, such as E2E time, since they exhibit similar trends. PG, data-driven methods (DeepDB, NeuroCard and FactorJoin) and our pretrained model exhibit a stable horizontal line since their performance is independent of the training workload. For the finetuned version of PRICE, we apply the corresponding training queries to tune the model further to fit each specific dataset. The results reported in Table 2 are obtained by the models finetuned (or trained) with 5×10^4 queries. Based on Table 2 and Figure 4, several key observations emerge:

1) PRICE can be further enhanced to achieve near-optimal performance by finetuning. In Table 2, our finetuned PRICE exhibits the smallest Q-ERROR and P-ERROR values across most quantiles. On most datasets, the P-ERROR values even approach 1.0, the lower bound of this metric. In terms of the E2E time, our finetuned PRICE achieves the shortest time in all four test datasets. The relative deviation to the optimal cases is all less than 0.4%.

2) The finetuning cost of PRICE is small and acceptable. From Figure 4, our PRICE could outperform almost all other methods in terms of Q-ERROR and P-ERROR when finetuning with even 100 queries. Collecting and executing such a few training queries takes less than one hour. The finetuning time to tune the model of PRICE is only around 10 minutes. Even if we collect and execute 5×10^4 queries by around 216 hours (or 9 days) and consume several hours to train models of MSCN and ALECE, their performance is still worse than PRICE. This is because the existing query-driven and hybrid CardEst methods need to be learned from scratch for each specific database. Whereas, for PRICE, the model has already

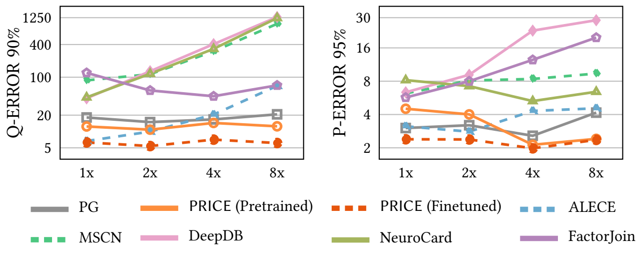


Figure 5: Estimation errors of cardinality estimation models during data updating processes.

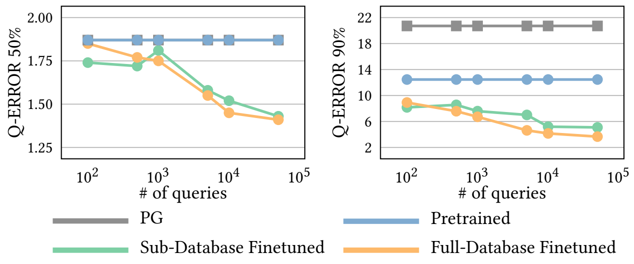


Figure 6: PRICE finetuned on the Sub-Database and the Full-Database workload and evaluated on the Full-Database.

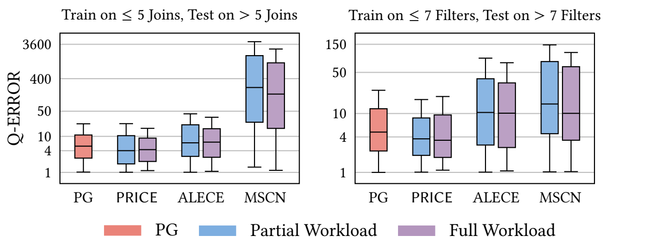
captured the general paradigm for mapping joint PDFs and query information to cardinality after pretraining, so we could tune the model to fit a specific database with a small number of queries.

6.4 Generalization Ability of PRICE

In this section, we examine the generalization ability of our pretrained model PRICE. We conduct experiments in three scenarios commonly occurring in real-world applications, namely data updates, data scaling and query workload drifts.

Generalization Ability to Data Updates. In real-world DBMS, data is constantly updated. We apply the STATS dataset with timestamps to test the performance of PRICE on updated data. Similar to [29, 47], we split the STATS datasets according to the creation time of tuples and obtained 4 datasets containing 1/8, 2/8, 4/8 and 8/8 tuples of the full data, respectively. We denote them as STATS 1x, 2x, 4x to 8x. For our PRICE, we directly test the pretrained model on the 4 datasets. For each existing CardEst method and the finetuned version of PRICE, we apply the underlying data and execute the query workload on STATS 1x (12.5% of the full data) for collecting workload statistics and model training (finetuning for PRICE). Then, we use the same test queries to evaluate its performance on STATS 1x, 2x, 4x to 8x. Notably, although the test queries remain the same on the 4 datasets, the true cardinalities differ due to data changes. We only report the results of 90%-quantile Q-ERROR and 95%-quantile P-ERROR and omit the similar results of other metrics.

From Figure 5, we find existing CardEst methods (except the basic PG) exhibit diminished performance when faced with data updates, and DeepDB, NeuroCard, and MSCN show notable performance declines. It also resembles their behaviors in Section 6.2 on unseen datasets. These models are specifically trained to fit each dataset, which may fall down on updated data with different distributions.



Q-ERROR 90%	Workloads	PG	PRICE	ALECE	MSCN
> 5 Joins	Partial	24.8	19.5(+4.2)	65.3(+21.4)	5170(+1868)
	Full		15.3 ↑	43.9 ↑	3302 ↑
> 7 Filters	Partial	31.7	15.5(-1.2)	129.6(+21.4)	1298(+453)
	Full		16.7 ↑	108.2 ↑	845 ↑

Figure 7: Estimation errors of cardinality estimation models across workload drifts.

However, both the original pretrained and finetuned PRICE maintain a promising and stable performance across all datasets. These results align with our observations in Section 6.2. The pretrained PRICE is trained to capture the meta-knowledge for CardEst, so it easily adapts to arbitrary datasets, including both unseen and updated data. For the finetuned PRICE, although we finetune it on STATS 1x, it could also adjust its parameters to capture some coarse-grained knowledge on the data distributions of the STATS dataset. As a result, when it is transferred to STATS 2x, 4x to 8x, the performance can still be further improved.

Generalization Ability to Data Scaling. Later, we present more details on the impact of different data volumes on the finetuned PRICE. We finetune PRICE on STATS 1x (partial of the dataset, denoted as Sub-Database) with a different number of queries and compare its performance with the model finetuned on the Full-Database (STATS 8x). Figure 6 exhibits a surprising result: the performance of PRICE finetuned on the partial and full dataset exhibits comparable performance. No matter how many training queries are applied, their performance is very close and consistently better than basic PG and original PRICE without finetuning. This further verifies the generalization ability of PRICE, which just focuses on the data distribution but not the data volume. This property is very appealing as executing the query workload on a smaller dataset is much faster, so we could consume much less time to collect statistics on a new dataset for finetuning. In the above setting, in comparison to executing the query workload on the full dataset for finetuning, we consume only 1.5% of time on STATS 1x.

Generalization Ability to Query Workload Drifts. In actual DBMS, the query workloads may change from time to time [74]. As data-driven CardEst methods are not sensitive to workload drifts [29, 87], we only compare our PRICE with ALECE and MSCN. For the training workload on a dataset, we obtain two partial workloads containing only no more than 5 joins and 7 filtering predicates, respectively. PRICE is pretrained on each partial workload derived from the 26 datasets (mentioned in Section 5) and tested on STATS. ALECE and MSCN are directly trained on each partial workload of STATS. For testing, we only employ queries from the test workload containing more than 5 joins or 7 filtering predicates to examine their performance on different queries. Figure 7 reports their performance on training (or pretraining for PRICE) with partial and full workloads. We have the following observations:

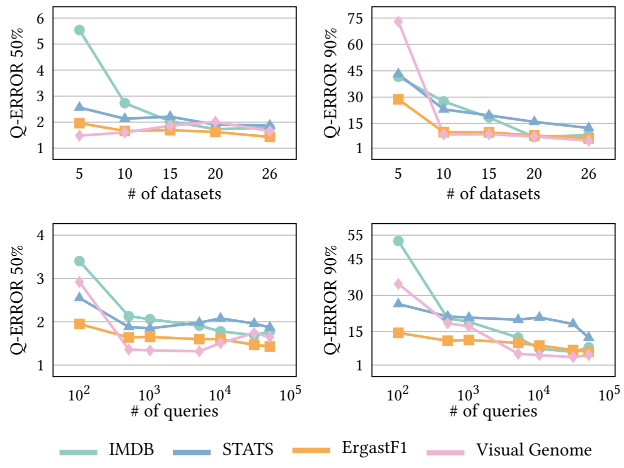


Figure 8: Evaluation of PRICE's performance with varied quantities of datasets and queries.

1) PRICE exhibits promising generalization ability to workload drifts. Despite training on the partial workload, PRICE shows minimal performance degradation (the blue numbers in Figure 7) compared to training on the full workload. This validates the effectiveness of the attention-based architecture in handling queries with different numbers of joins and filtering predicates. Similarly, another attention-based approach, ALECE, also maintains relatively stable performance. Its Q-ERROR remains consistent, but is much higher than PRICE, indicating the potential limitations in handling complex query patterns. For MSCN, which uses simple deep neural networks, the performance drops very significantly.

2) For complex queries with a relatively large number of joins or filtering predicates, only our PRICE could obtain better performance than the basic PG. On the contrary, both ALECE and MSCN perform very poorly on such complex queries (consistently worse than PG), no matter whether they are trained on full or partial workloads. This is due to the different learning mechanisms between PRICE and existing methods. PRICE aims to learn the general approach to combine and correct the simple distributions on every single table (encoded as features) to the joint PDF, making it easily applicable to any number of joins and filtering predicates. Whereas, the models in ALECE and MSCN learn the specific parameters to map queries to their cardinality. As the training workload may not contain a sufficient number of complex queries (executing them is time-consuming), the models may not be well trained to capture such mapping relations.

6.5 Impacts of Training Data to Pretrain PRICE

Collecting broader and richer training data from varied databases is crucial for developing a robust pretrained model. In this set of experiments, we explore how the quantity and diversity of datasets and the volume of training workload impact our pretrained PRICE.

Number of Training Datasets. Figure 8 shows PRICE's performance on unseen datasets that are pretrained using different numbers of datasets. We find that:

1) There is a clear trend that the 50% and 90%-quantile Q-ERROR of PRICE progressively improves when the number of datasets increases and eventually stabilizes after around 15—20 datasets. The

trends of other metrics are also similar, so we omit them due to space limits. It indicates that PRICE can capture enough knowledge and transferable parameters for CardEst using not too many datasets.

2) With a small number of datasets, the performance of PRICE varies significantly in different datasets. Specifically, when trained using only 5 datasets, PRICE performs well on ErgastF1 but poorly on IMDB. However, when trained over 15 datasets, its performance remains stable over all datasets (also witnessed in Section 6.2). This is because a small number of datasets are not diversified enough, so the knowledge learned by PRICE may be biased and can not be generalized well. This underscores the importance of selecting datasets from a variety of domains with diverse properties to achieve the high generalization capability of PRICE.

Number of Training Queries. Beyond the dataset quantity, the number of training queries collected from each dataset also directly affects the performance of the pretrained PRICE. Therefore, we also pretrain PRICE with different numbers of queries on each dataset and evaluate its performance on four unseen test datasets.

From Figure 8, we observe that the overall performance of PRICE steadily improves when more queries are applied for training. It is natural as more training queries bring the PRICE's model more knowledge on how to filter the joint PDF for computing cardinality. Additionally, the improvement of PRICE becomes marginal after 1.0×10^3 training queries, indicating the model has mastered enough knowledge. At that time, the workload contains only 2.6×10^4 queries to pretrain PRICE. It is much cheaper than existing query-driven and hybrid CardEst methods, which require collecting a large volume of training queries on each dataset. We note that, on the STATS dataset, ALECE is required to collect around the same volume of queries to attain comparable performance of PRICE pretrained with 2.6×10^4 queries. For MSCN, even if we train it on 5×10^4 queries (results reported in Table 2), its performance is still much worse than PRICE pretrained with only 2.6×10^4 queries.

M2
M5
R1W3
R3D5

6.6 Ablation Study

We systematically assessed the impact on model performance of each feature category outlined in Section 4 on four test datasets: the value distribution of each attribute, the scaling factor distribution of each join condition, the information of each filtering condition and table-level and query-level auxiliary information. Figure 9 highlights the effects of removing each feature category. Comparing with the original pretrained PRICE, we find that:

1) Removing any single category of features results in varying degrees of performance degradation. This verifies the effectiveness of our feature selection strategy in PRICE.

2) Excluding the value distribution of each attribute leads to a significant decline in model accuracy. As discussed earlier, value distributions serve as fundamental tokens in our approach, enabling PRICE to effectively learn correlations between different tables and attributes. Without this feature, PRICE can not capture useful information w.r.t. the joint PDF for CardEst.

3) Excluding table-level and query-level information leads to significant performance degradation. This information, including table size and cardinality estimates from traditional methods, is crucial for PRICE to approximate cardinality scales. While other features are normalized to $[0, 1]$ for comparability across different databases, the differences in database sizes necessitate such

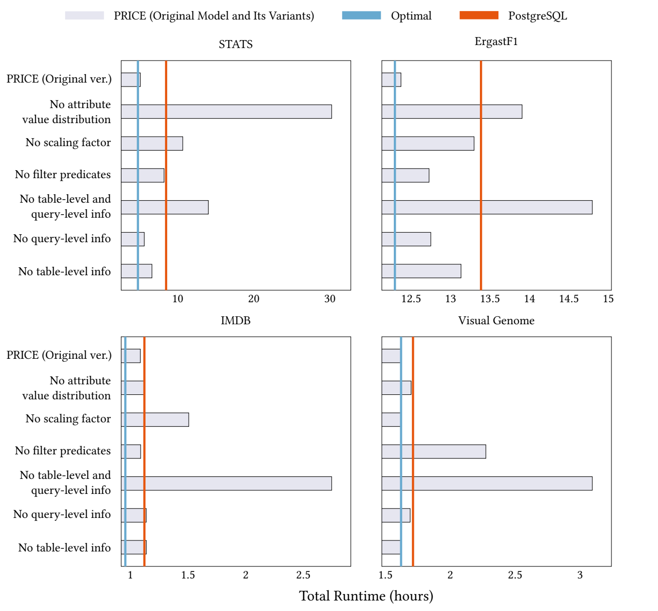


Figure 9: Ablation study. Each label (y-axis) is a difference from the PRICE.

guidelines. Without them, PRICE struggles to accurately position its estimates.

To further investigate this, we also conducted experiments where we individually removed table-level and query-level information. When either category was excluded, the remaining feature (whether table size or traditional cardinality estimates) still provided sufficient guidance for the model to estimate cardinality magnitudes. The results demonstrate that retaining at least one of these categories significantly enhances model performance, thus supporting our hypothesis. Furthermore, the removal of either table-level or query-level information results in performance degradation, demonstrating that each type of auxiliary information contributes to the model from different perspectives, which enhances robustness.

7 CONCLUSION AND FUTURE WORK

In this paper, we introduce PRICE, a PRetrained multI-table CardEst model that overcomes the limitations of existing methods by enabling quick deployment to new, unseen databases with minimal preparation. After pretraining on diverse datasets, PRICE shows strong generalization and outperforms existing approaches. After finetuning with minimal database-specific data, it achieves near-optimal performance while incurring lower time and space costs, making it a practical solution for DBMS.

However, PRICE has some limitations: it is primarily designed for OLAP scenarios, resulting in slower inference times compared to traditional methods. It currently supports only basic SPJ queries and lacks support for complex query types like non-equal joins or nested queries. Additionally, its performance relies on the diversity of pretraining data, potentially limiting generalization to rare or extreme cases. For future enhancements of PRICE, it is crucial to extend its capabilities to handle more complex query types. Meanwhile, we plan to extend the pretraining paradigm to other DBMS tasks, including but not limited to cost estimation, index recommendation, and view advisor.

M5
R1D5
R3D6

REFERENCES

[1] ALECE Github repository. https://github.com/pfl-cs/ALECE.

[2] DeepDB Github repository. https://github.com/DataManagementLab/deepdb-public.

[3] ErgastF1 Dataset. https://relational-data.org/dataset/ErgastF1.

[4] FactorJoin Github repository. https://github.com/wuziniu/FactorJoin.

[5] Full Version paper of PRICE. https://github.com/StCarmen/PRICE/blob/master/PRICE_full_paper.pdf.

[6] IMDB Dataset. http://homepages.cwi.nl/~boncz/job/imdb.tgz.

[7] Job-light Workload. https://github.com/andreaskipf/learnedcardinalities/blob/master/workloads/job-light.sql.

[8] MSCN Github repository. https://github.com/andreaskipf/learnedcardinalities.

[9] NeuroCard Github repository. https://github.com/neurocard/neurocard.

[10] online. https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp.

[11] online. https://stats.stackexchange.com/.

[12] PRICE Github repository. https://github.com/StCarmen/PRICE.

[13] STATS Dataset. https://relational.fit.cvut.cz/dataset/Stats.

[14] VisualGenome Dataset. https://relational-data.org/dataset/VisualGenome.

[15] Pratyush Agnihotri, Boris Koldehofe, Carsten Binnig, and Manisha Luthra. 2023. Zero-Shot Cost Models for Parallel Stream Processing. In *Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. ACM, Seattle WA USA, 1–5. https://doi.org/10.1145/3593078.3593934

[16] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.

[17] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2001. STHoles: A Multidimensional Workload-Aware Histogram. In *SIGMOD*. 211–222.

[18] C. K. Chow and C. N. Liu. 1968. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theory* 14, 3 (1968), 462–467.

[19] Amol Deshpande, Minos Garofalakis, and Rajeev Rastogi. 2001. Independence is good: Dependency-based histogram synopses for high-dimensional data. *ACM SIGMOD Record* 30, 2 (2001), 199–210.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[21] Postgresql Documentation 12. 2020. Chapter 70.1. Row Estimation Examples. *https://www.postgresql.org/docs/current/row-estimation-examples.html* (2020).

[22] Anshuman Dutt, Chi Wang, Vivek R. Narasayya, and Surajit Chaudhuri. 2020. Efficiently Approximating Selectivity Functions using Low Overhead Regression Models. *Proc. VLDB Endow.* 13, 11 (2020), 2215–2228. http://www.vldb.org/pvldb/vol13/p2215-dutt.pdf

[23] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates using Lightweight Models. *Proc. VLDB Endow.* 12, 9 (2019), 1044–1057.

[24] Dennis Fuchs, Zhen He, and Byung Suk Lee. 2007. Compressed histograms with arbitrary bucket layouts for selectivity estimation. *Inf. Sci.* 177, 3 (2007), 680–702.

[25] Lise Getoor, Benjamin Taskar, and Daphne Koller. 2001. Selectivity estimation using probabilistic models. In *SIGMOD*. 461–472.

[26] PostgreSQL Global Development Group. 1996. PostgreSQL. https://www.postgresql.org. (1996). Accessed: 2022-10-28.

[27] Dimitrios Gunopulos, George Kollios, Vassilis J Tsotras, and Carlotta Domeniconi. 2000. Approximating multi-dimensional aggregate range queries over real attributes. In *SIGMOD*. 463–474.

[28] Dimitrios Gunopulos, George Kollios, Vassilis J Tsotras, and Carlotta Domeniconi. 2005. Selectivity estimators for multidimensional range queries over real attributes. *The VLDB Journal* 14, 2 (2005), 137–154.

[29] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proc. VLDB Endow.* 15, 4 (2021), 752–765.

[30] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2019. Multi-attribute selectivity estimation using deep learning. In *SIGMOD*.

[31] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer.

[32] Max Heimel, Martin Kiefer, and Volker Markl. 2015. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*. 1477–1492.

[33] Roman Heinrich, Manisha Luthra, Harald Kornmayer, and Carsten Binnig. 2022. Zero-shot cost models for distributed stream processing. In *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*. ACM, Copenhagen Denmark, 85–90. https://doi.org/10.1145/3524860.3539639

[34] Benjamin Hilprecht and Carsten Binnig. 2022. One Model to Rule them All: Towards Zero-Shot Learning for Databases. http://arxiv.org/abs/2105.00642 arXiv:2105.00642 [cs].

[35] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-shot cost models for out-of-the-box learned cost prediction. *Proceedings of the VLDB Endowment* 15, 11 (July 2022), 2361–2374. https://doi.org/10.14778/3551793.3551799

[36] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *Proc. VLDB Endow.* 13, 7 (2020), 992–1005.

[37] Andranik Khachatryan, Emmanuel Müller, Christian Stier, and Klemens Böhm. 2015. Improving Accuracy and Robustness of Self-Tuning Histograms by Subspace Clustering. *IEEE Trans. Knowl. Data Eng.* 27, 9 (2015), 2377–2389.

[38] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models. *Proc. VLDB Endow.* 10, 13 (2017), 2085–2096.

[39] Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. 2022. Learned Cardinality Estimation: An In-depth Study. In *Proceedings of the 2022 International Conference on Management of Data*. ACM, Philadelphia PA USA, 1214–1227. https://doi.org/10.1145/3514221.3526154

[40] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[41] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*.

[42] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *NIPS*. 6402–6413.

[43] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. 2018. Deep Neural Networks as Gaussian Processes. In *ICLR*.

[44] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *PVLDB* 9, 3 (2015), 204–215.

[45] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. 2017. Cardinality Estimation Done Right: Index-Based Join Sampling. In *CIDR*.

[46] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander join: Online aggregation via random walks. In *SIGMOD*. 615–629.

[47] Pengfei Li, Wenqing Wei, Rong Zhu, Bolin Ding, Jingren Zhou, and Hua Lu. 2023. ALECE: An Attention-based Learned Cardinality Estimator for SPJ Queries on Dynamic Workloads. *Proceedings of the VLDB Endowment* 17, 2 (Oct. 2023), 197–210. https://doi.org/10.14778/3626292.3626302

[48] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality estimation using neural networks. In *Proceedings of 25th Annual International Conference on Computer Science and Software Engineering, CASCON 2015, Markham, Ontario, Canada, 2-4 November, 2015*. IBM / ACM, 53–59. http://dl.acm.org/citation.cfm?id=2886453

[49] Jie Liu, Wenqian Dong, Dong Li, and Qingqing Zhou. 2021. Fauce: Fast and Accurate Deep Ensembles with Uncertainty for Cardinality Estimation. *Proc. VLDB Endow.* 14, 11 (2021), 1950–1963.

[50] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[51] Pedro Lopes, Craig Guyer, and Milener Gene. 2019. Sql docs: cardinality estimation (SQL Server). *https://docs.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver15* (2019).

[52] David Lopez-Paz, Philipp Hennig, and Bernhard Schölkopf. [n.d.]. The Randomized Dependence Coefficient. ([n. d.]).

[53] Yao Lu, Srikanth Kandula, Arnd Christian König, and Surajit Chaudhuri. 2021. Pre-training summarization models of structured datasets for cardinality estimation. *Proceedings of the VLDB Endowment* 15, 3 (Nov. 2021), 414–426. https://doi.org/10.14778/3494124.3494127

[54] MariaDB Server Documentation. 2020. Statistics for optimizing queries: InnoDB persistent statistics. https://mariadb.com/kb/en/innodb-persistent-statistics/. Accessed: 2020.

[55] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2006. An integrated efficient solution for computing frequent and top-k elements in data streams.

ACM Trans. Database Syst. 31, 3 (sep 2006), 1095–1133. https://doi.org/10.1145/1166074.1166084

[56] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1301.3781

[57] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *Proc. VLDB Endow.* 2, 1 (2009), 982–993.

[58] Jan Motl and Oliver Schulte. 2015. The CTU Prague Relational Learning Repository. *CoRR* abs/1511.03086 (2015). arXiv:1511.03086 http://arxiv.org/abs/1511.03086

[59] M Muralikrishna and David J DeWitt. 1988. Equi-depth multidimensional histograms. In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data.* 28–36.

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

[61] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 237–252.

[62] Oracle White Paper. 2019. The optimizer In Oracle database 19c. *https://www.oracle.com/technetwork/database/bi-datawarehousing/twp-optimizer-with-oracledb-19c-5324206.pdf* (2019).

[63] Hoifung Poon and Pedro M. Domingos. 2011. Sum-Product Networks: A New Deep Architecture. In *UAI.* 337–346.

[64] Viswanath Poosala and Yannis E Ioannidis. 1997. Selectivity estimation without the attribute value independence assumption. In *VLDB*, Vol. 97. 486–495.

[65] MySQL 8.0 Reference Manual. 2020. Chapter 15.8.10.2 Configuring Non-Persistent Optimizer Statistics Parameters. *https://dev.mysql.com/doc/refman/8.0/en/innodb-statistics-estimation.html* (2020).

[66] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1979. Access path selection in a relational database management system. In *SIGMOD.* 23–34.

[67] MariaDB Server Documentation. 2020. Statistics for optimizing queries: InnoDB persistent statistics. *https://mariadb.com/kb/en/innodb-persistent-statistics/* (2020).

[68] Utkarsh Srivastava, Peter J. Haas, Volker Markl, Marcel Kutsch, and Tam Minh Tran. 2006. ISOMER: Consistent Histogram Construction Using Query Feedback. In *ICDE.* 39.

[69] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO - DB2's LEarning Optimizer. In *VLDB.* 19–28.

[70] Ji Sun, Jintao Zhang, Zhaoyan Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation: A Design Space Exploration and A Comparative Evaluation. *Proc. VLDB Endow.* 15, 1 (2021), 85–97.

[71] Kostas Tzoumas, Amol Deshpande, and Christian S Jensen. 2011. Lightweight graphical models for selectivity estimation without independence assumptions. *PVLDB* 4, 11 (2011), 852–863.

[72] 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 *NeurIPS.* 5998–6008.

[73] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *VLDB* 14, 9 (2021), 1640–1654.

[74] Lianggui Weng, Rong Zhu, Di Wu, Bolin Ding, Bolong Zheng, and Jingren Zhou. 2024. Eraser: Eliminating Performance Regression on Learned Query Optimizer. *Proc. VLDB Endow.* 17, 5 (may 2024), 926–938. https://doi.org/10.14778/3641204.3641205

[75] Chenggang Wu, Alekh Jindal, Saeed Amizadeh, Hiren Patel, Wangchao Le, Shi Qiao, and Sriram Rao. 2018. Towards a Learning Optimizer for Shared Clouds. *Proc. VLDB Endow.* 12, 3 (2018), 210–222.

[76] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation. In *SIGMOD.* 2009–2022.

[77] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2023. FactorJoin: A New Cardinality Estimation Framework for Join Queries. *Proc. ACM Manag. Data* 1, 1 (2023), 41:1–41:27.

[78] Ziniu Wu and Amir Shaikhha. 2020. BayesCard: A Unified Bayesian Framework for Cardinality Estimation. *CoRR* abs/2012.14743 (2020). https://arxiv.org/abs/2012.14743

[79] Ziniu Wu, Pei Yu, Peilun Yang, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2022. A Unified Transferable Model for ML-Enhanced DBMS. In *CIDR.*

[80] Ziniu Wu, Rong Zhu, Andreas Pfadler, Yuxing Han, Jiangneng Li, Zhengping Qian, Kai Zeng, and Jingren Zhou. 2020. FSPN: A New Class of Probabilistic Graphical Model. *CoRR* abs/2011.09020 (2020). https://arxiv.org/abs/2011.09020

[81] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *Proc. VLDB Endow.* 14, 1 (2020), 61–73.

[82] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *Proc. VLDB Endow.* 13, 3 (2019), 279–292.

[83] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do Transformers Really Perform Badly for Graph Representation?. In *NeurIPS.* 28877–28888.

[84] Kangfei Zhao, Jeffrey Xu Yu, Zongyan He, Rui Li, and Hao Zhang. 2022. Lightweight and Accurate Cardinality Estimation by Neural Network Gaussian Process. In *SIGMOD.* 973–987.

[85] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random sampling over joins revisited. In *SIGMOD.* 1525–1539.

[86] Rong Zhu, Lianggui Weng, Wenqing Wei, Di Wu, Jiazhen Peng, Yifan Wang, Bolin Ding, Defu Lian, Bolong Zheng, and Jingren Zhou. 2024. PilotScope: Steering Databases with Machine Learning Drivers. *Proc. VLDB Endow.* 17, 5 (may 2024), 980–993. https://doi.org/10.14778/3641204.3641209

[87] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *Proc. VLDB Endow.* 14, 9 (2021), 1489–1502.

Appendix

Due to space limitations, we present some minute details and relatively less important experimental results and analyses in this appendix.

A DETAILED EXPLANATION OF THE SCALING FACTORS

M5
R1W2
R3D1

Below, we offer a detailed explanation of the computation process, supported by supplementary experiments, which demonstrate that the overhead for maintaining this information is acceptable.

For each database, we compute the distribution of scaling factors for all potential join conditions offline and in parallel. This process is efficient and incurs minimal computational time. (see Part A ►)

Then, we show that the computation, as well as the updating, of the distribution vectors of scaling factors could be done in a more efficient way using sampling and periodical update strategies. (see Parts B and C ►)

(a) Table T_A

A_1	A_2	$S_{A,B}$
b	0	1
c	2	1
b	3	2
c	4	1

(b) Table T_B

B_1	B_2	B_3	$S_{A,B}$
1	0.32	M	0
0	0.67	D	1
3	0.41	D	1
2	0.79	M	1
4	0.58	K	1
3	0.21	K	1

(c) Table $T_A \rtimes\!\!\!\ltimes T_B$

A_1	A_2	B_1	B_2	B_3
null	null	1	0.32	M
b	0	0	0.67	D
b	3	3	0.41	D
c	2	2	0.79	M
c	4	4	0.58	K
b	3	3	0.21	K

Figure 10: Example of Scaling Factor Calculation.

Part A. Offline Computation for the Scaling Factor Distribution. To clarify how the scaling factor is derived, consider the example depicted in Figure 10. Given the join condition $T_A.A_2 = T_B.B_1$, We calculate the scaling factors $S_{A,B}$ and $S_{B,A}$ to quantify the scaling effect on entities in Table T_A and Table T_B after the join operation. For instance, $S_{A,B}$ indicates how many records in Table T_B can join with this record in Table T_A and vice versa. This calculation method is detailed in Section 5 of our previous work[87]. After obtaining the scaling factors (e.g., $S_{A,B}$ and $S_{B,A}$), we compute the histogram of these factors to describe their distribution.

This process only requires scanning the two tables involved in the join, resulting in minimal time complexity. Based on our experiments on real-world datasets, calculating the scaling factors for a join condition over two tables with millions of tuples typically takes less than 0.5 seconds. This computation can be executed in the background without disrupting the online operation of the database. Furthermore, since the computation for different tables is independent, it can be parallelized to further reduce the computational burden. Once calculated, these statistics can be reused for all future queries involving the same join condition, thus amortizing the computational cost over time.

Part B. Sampling Strategy for Large Tables is Feasible. Notice that our model only requires the distribution of the scaling factors, not the precise values. Thus, for super-large tables involved in the join condition, we could sample a fraction of tuples to estimate the scaling factor distribution. We conduct an experiment on the largest dataset IMDB (more than 4.5GB) used in our evaluation with sampling rates of 10%, 1%, 0.1%, and 0.05%. The results, as illustrated in Table 3, demonstrate that sampling effectively provides the necessary information for the model and maintains stable performance without significant degradation. Our model could still yield satisfactory results even when we sample a very small fraction (1/2000) of tuples for computing the distribution of scaling factors.

Table 3: The performance of PRICE as we change the sample rate of the data to generate the scaling factor distribution.

Sample Rate	Q-ERROR			
	50%	90%	95%	99%
Full Table	1.78	8.40	15.45	70.89
10.0%	1.77	8.18	15.49	69.70
1.00%	1.78	8.60	16.17	71.07
0.10%	1.78	8.54	15.56	67.82
0.05%	1.72	8.42	14.88	61.48

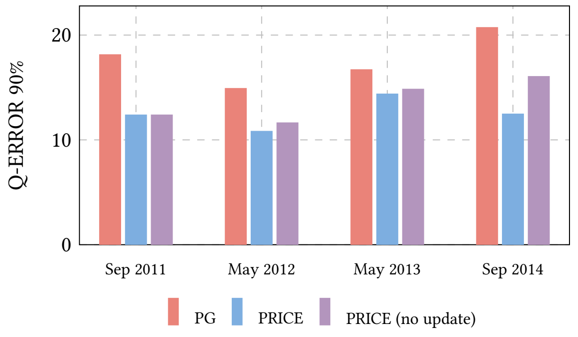


Figure 11: Performance of PRICE over time without updating scaling factor distribution.

Part C. Periodic Updating of Scaling Factors is Acceptable. Real-time updates of scaling factor distributions may be challenging; thus, we advocate for periodic updates during practical deployment. These updates can be executed in the background, ensuring no disruption to real-time query responses. In practical scenarios, changes in the scaling factor distributions are typically not drastic. Our experiments (see below) also indicate that models could tolerate the staleness of the distribution information to some extent. Thus, periodic updating could sustain model performance effectively.

In our experiments, we apply the real-world STATS dataset, covering data from July 2010 to September 2014, we use the distribution of scaling factors computed over data before September 2011 as the model input and do not update them further. We evaluate the performance of our PRICE at subsequent timestamps: May 2012, May 2013 and September 2014 and show the result in Figure 11. Although the model's performance declines slightly over time without updating the scaling factors, it still outperforms PG even without updating the scaling factor information for the last three years (from September 2011 to September 2014, see the last timestamp in the Figure 11). Thus, we could update the scaling factor information with reasonable frequency to balance the time cost and model accuracy.

M1

R1W1

R2W1

R2D2

B EXAMPLES OF PRICE'S ATTENTION MAPS

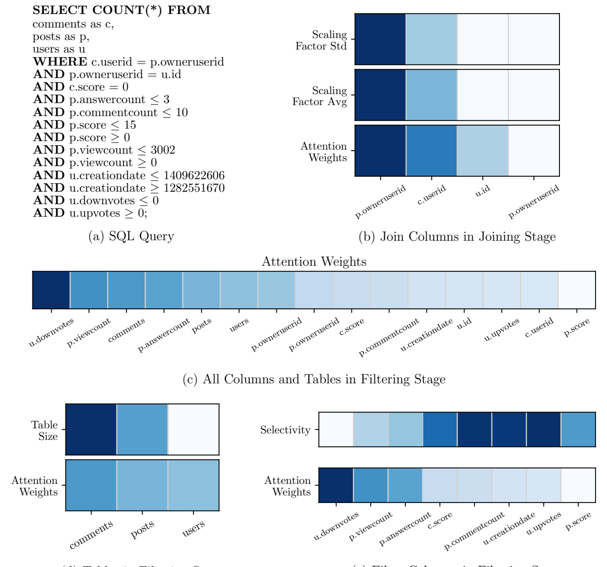
We provide some examples of PRICE's attention maps to support the analysis presented in Section 4. These examples, drawn from various datasets and queries, reveal a consistent pattern that confirms our hypotheses about the functionality of PRICE's attention mechanism.

As shown in Figure 12, even across different queries, PRICE's attention mechanism adaptively assigns weights to different tables and attributes based on their relevance to the final cardinality. Specifically, we observe the following phenomena:

(1) In the joining stage, join attributes with larger mean scaling factors or more variable scaling factors (i.e., greater standard deviations) receive higher attention weights;
(2) In the filtering stage, the model primarily focuses on attributes with filtering conditions;
(3) PRICE tends to allocate more attention to larger tables;
(4) PRICE focuses more on filtering attributes with lower selectivity.

```
SELECT COUNT(*) FROM
comments as c,
posts as p,
users as u
WHERE c.userid = p.owneruserid
AND p.owneruserid = u.id
AND c.score = 0
AND p.answercount ≤ 3
AND p.commentcount ≤ 10
AND p.score ≤ 15
AND p.score ≥ 0
AND p.viewcount ≤ 3002
AND p.viewcount ≥ 0
AND u.creationdate ≤ 1409622606
AND u.creationdate ≥ 1282551670
AND u.downvotes ≤ 0
AND u.upvotes ≥ 0;
```

(a) SQL Query



(b) Join Columns in Joining Stage

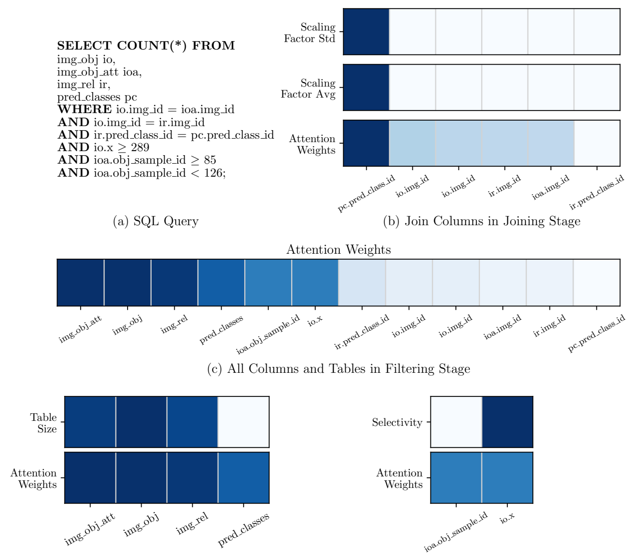
(c) All Columns and Tables in Filtering Stage

(d) Tables in Filtering Stage

(e) Filter Columns in Filtering Stage

```
SELECT COUNT(*) FROM
img_obj io,
img_obj_att ioa,
img_rel ir,
pred_classes pc
WHERE io.img_id = ioa.img_id
AND io.img_id = ir.img_id
AND ir.pred_class_id = pc.pred_class_id
AND io.x ≥ 289
AND ioa.obj_sample_id ≥ 85
AND ioa.obj_sample_id < 126;
```

(a) SQL Query



(b) Join Columns in Joining Stage

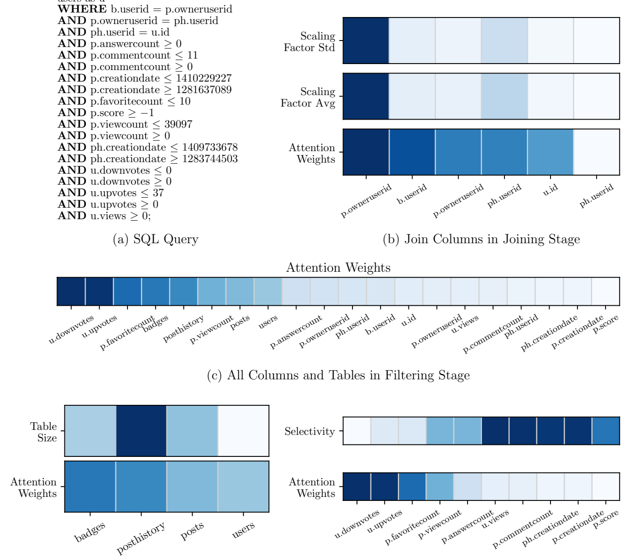
(c) All Columns and Tables in Filtering Stage

(d) Tables in Filtering Stage

(e) Filter Columns in Filtering Stage

```
SELECT COUNT(*) FROM
badges as b,
posthistory as ph,
posts as p,
users as u
WHERE b.userid = p.owneruserid
AND p.owneruserid = ph.userid
AND ph.userid = u.id
AND p.answercount ≥ 0
AND p.commentcount ≤ 11
AND p.commentcount ≥ 0
AND p.creationdate ≤ 1410229227
AND p.creationdate ≥ 1281637089
AND p.favoritecount ≤ 10
AND p.score ≥ −1
AND p.viewcount ≤ 39097
AND p.viewcount ≥ 0
AND ph.creationdate ≤ 1409733678
AND ph.creationdate ≥ 1283744503
AND u.downvotes ≤ 0
AND u.downvotes ≥ 0
AND u.upvotes ≤ 37
AND u.upvotes ≥ 0
AND u.views ≥ 0;
```

(a) SQL Query



(b) Join Columns in Joining Stage

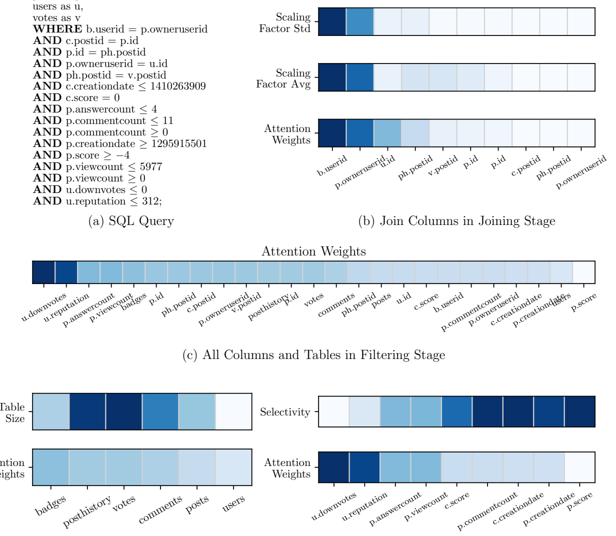
(c) All Columns and Tables in Filtering Stage

(d) Tables in Filtering Stage

(e) Filter Columns in Filtering Stage

```
SELECT COUNT(*) FROM
badges as b,
comments as c,
posthistory as ph,
posts as p,
users as u,
votes as v
WHERE b.userid = p.owneruserid
AND c.postid = p.id
AND p.id = ph.postid
AND p.owneruserid = u.id
AND ph.postid = v.postid
AND c.creationdate ≤ 1410263909
AND c.score = 0
AND p.answercount ≤ 4
AND p.commentcount ≤ 11
AND p.commentcount ≥ 0
AND p.creationdate ≥ 1295915501
AND p.score ≥ −4
AND p.viewcount ≤ 5977
AND p.viewcount ≥ 0
AND u.downvotes ≤ 0
AND u.reputation ≤ 312;
```

(a) SQL Query



(b) Join Columns in Joining Stage

(c) All Columns and Tables in Filtering Stage

(d) Tables in Filtering Stage

(e) Filter Columns in Filtering Stage

Figure 12: Some examples of PRICE's attention map.

Table 4: Overview of datasets: database statistics and workload specifications.

Dataset Name	Database Statistics										Workload Specifications			
	Sectors	# of Tables	# of Cols (All Tables)	# of Rows (All Tables)	# of Join Relations	Volume	Total Attribute Domain Size	Distribution Skewness (MIN/AVG/MAX)	Average Pairwise Correlation	Join Forms	Joined Tables	# of Filtering Predicates	Join Type	True Cardinality Range
Accidents	Government	3	43	$1.5 \cdot 10^6$	3	118.6M	$1.8 \cdot 10^6$	-0.4/1.3/8.1	0.33	Chain	2 - 3	0 - 7	PK-FK/FK-FK	10^1 - $6{\cdot}10^9$
Airline	Retail	19	119	$1.1 \cdot 10^7$	22	4.5G	$1.2 \cdot 10^5$	-2.2/5.1/129.8	0.17	Star	2 - 7	2 - 7	PK-FK	10^1 - $1{\cdot}10^7$
Baseball	Sport	25	359	$4.7 \cdot 10^5$	103	30.6M	$3.1 \cdot 10^5$	-13.6/1.7/101.7	0.38	Mixed	2 - 7	0 - 7	PK-FK/FK-FK	10^1 - $6{\cdot}10^9$
Basketball	Sport	9	198	$4.5 \cdot 10^4$	32	4.8M	$1.2 \cdot 10^5$	-4.4/2.6/90.0	0.53	Mixed	2 - 7	0 - 6	PK-FK/FK-FK	10^1 - $1{\cdot}10^{10}$
Carcinogenesis	Medicine	6	24	$2.8 \cdot 10^4$	15	776.0K	$5.3 \cdot 10^4$	-0.2/0.0/0.2	0.69	Star	2 - 6	0 - 6	PK-FK/FK-FK	10^1 - $5{\cdot}10^7$
CCS	Financial	5	21	$4.2 \cdot 10^5$	5	8.1M	$3.5 \cdot 10^5$	-1.6/2.2/18.5	0.35	Mixed	2 - 5	0 - 10	PK-FK/FK-FK	10^1 - $4{\cdot}10^5$
ChEMBL	Medicine	9	80	$3.7 \cdot 10^6$	19	1.7G	$8.4 \cdot 10^6$	-12.7/93.6/1341.4	0.52	Mixed	2 - 7	0 - 10	PK-FK/FK-FK	10^1 - $2{\cdot}10^9$
Consumer	Retail	3	25	$2.2 \cdot 10^6$	3	78.9M	$3.0 \cdot 10^6$	-1.6/5.9/106.4	0.40	Chain	2 - 3	1 - 6	PK-FK/FK-FK	10^1 - $6{\cdot}10^6$
Credit	Synthetic	8	73	$1.6 \cdot 10^6$	13	82.0M	$1.8 \cdot 10^6$	-1.8/0.0/2.3	0.21	Mixed	2 - 7	0 - 6	PK-FK/FK-FK	10^1 - $3{\cdot}10^9$
Employee	Synthetic	6	25	$3.9 \cdot 10^6$	11	134.1M	$1.3 \cdot 10^6$	-0.2/0.2/0.8	0.04	Mixed	2 - 6	0 - 3	PK-FK/FK-FK	10^1 - $5{\cdot}10^6$
Financial	Financial	8	55	$1.1 \cdot 10^6$	11	65.1M	$1.3 \cdot 10^6$	-0.1/1.4/7.6	0.37	Mixed	2 - 7	0 - 7	PK-FK/FK-FK	10^1 - $3{\cdot}10^8$
FNHK	Medicine	3	24	$2.1 \cdot 10^6$	3	68.6M	$1.9 \cdot 10^5$	-1.0/8.6/46.2	0.24	Chain	2 - 3	0 - 7	PK-FK/FK-FK	10^1 - $6{\cdot}10^7$
Grants	Education	12	51	$3.0 \cdot 10^6$	21	701.0M	$3.6 \cdot 10^6$	-1.2/25.5/340.3	0.22	Mixed	2 - 7	0 - 10	PK-FK/FK-FK	10^1 - $2{\cdot}10^6$
Hepatitis	Medicine	7	26	$1.3 \cdot 10^4$	9	224.0K	$1.4 \cdot 10^4$	-0.2/0.1/0.8	0.25	Mixed	2 - 7	0 - 7	PK-FK/FK-FK	10^1 - $1{\cdot}10^4$
Hockey	Sport	18	300	$9.5 \cdot 10^4$	145	9.4M	$9.0 \cdot 10^4$	-1.9/0.5/5.6	0.33	Mixed	2 - 7	0 - 9	PK-FK/FK-FK	10^1 - $2{\cdot}10^7$
LegalActs	Government	5	33	$1.8 \cdot 10^6$	8	188.8M	$2.0 \cdot 10^6$	-15.3/-0.3/5.4	0.22	Mixed	2 - 4	0 - 10	PK-FK/FK-FK	10^1 - $1{\cdot}10^6$
MovieLens	Entertainment	7	24	$1.2 \cdot 10^6$	9	17.8M	$2.1 \cdot 10^5$	-1.7/-0.4/0.1	0.12	Mixed	2 - 7	0 - 7	PK-FK/FK-FK	10^1 - $5{\cdot}10^7$
Sakila	Synthetic	15	86	$4.6 \cdot 10^4$	31	2.9M	$1.2 \cdot 10^5$	-6.0/-0.1/0.5	0.18	Mixed	2 - 7	0 - 10	PK-FK/FK-FK	10^1 - $3{\cdot}10^{10}$
SAP	Synthetic	5	45	$4.1 \cdot 10^6$	6	172.7M	$4.6 \cdot 10^6$	-0.8/-0.1/1.0	0.54	Star	2 - 4	0 - 10	PK-FK/FK-FK	10^1 - $3{\cdot}10^6$
Seznam	Retail	4	14	$2.7 \cdot 10^6$	6	67.4M	$3.1 \cdot 10^5$	0.7/22.0/101.1	0.19	Star	2 - 4	0 - 7	PK-FK/FK-FK	10^1 - $3{\cdot}10^8$
SSB	Synthetic	5	58	$1.3 \cdot 10^7$	4	1.2G	$1.2 \cdot 10^7$	-5.3/0.1/5.8	0.13	Star	2 - 5	0 - 7	PK-FK	10^1 - $1{\cdot}10^7$
TalkingData	Technology	8	25	$6.9 \cdot 10^7$	10	2.1G	$7.5 \cdot 10^6$	-1.4/0.0/1.4	0.65	Mixed	2 - 7	0 - 10	PK-FK/FK-FK	10^1 - $8{\cdot}10^9$
Telstra	Government	5	12	$1.5 \cdot 10^5$	10	2.9M	$9.4 \cdot 10^4$	0.0/1.8/11.6	0.05	Star	2 - 5	0 - 5	PK-FK/FK-FK	10^1 - $2{\cdot}10^5$
Tournament	Sport	9	106	$2.0 \cdot 10^5$	21	9.6M	$7.1 \cdot 10^3$	-0.4/0.6/7.2	0.17	Mixed	2 - 7	0 - 7	PK-FK/FK-FK	10^1 - $2{\cdot}10^9$
TPC-H	Synthetic	8	61	$8.7 \cdot 10^6$	12	1.1G	$1.5 \cdot 10^7$	0.0/0.1/0.6	0.08	Mixed	2 - 7	0 - 7	PK-FK/FK-FK	10^1 - $5{\cdot}10^9$
TubePricing	Education	6	40	$1.5 \cdot 10^5$	9	5.7M	$1.9 \cdot 10^5$	-1.9/17.5/240.2	0.51	Mixed	2 - 6	0 - 10	PK-FK/FK-FK	10^1 - $1{\cdot}10^5$
IMDB	Entertainment	6	37	$6.2 \cdot 10^7$	15	3.0G	$8.5 \cdot 10^7$	-1.9/19.2/384.5	0.32	Star	2 - 5	0 - 4	PK-FK/FK-FK	2 - $1{\cdot}10^{10}$
STATS	Education	8	43	$1.0 \cdot 10^6$	30	32.3M	$1.9 \cdot 10^6$	-1.0/10.5/134.5	0.46	Mixed	2 - 7	0 - 10	PK-FK/FK-FK	16 - $8{\cdot}10^{10}$
ErgastF1	Sport	14	98	$5.5 \cdot 10^5$	67	15.1M	$3.2 \cdot 10^5$	-2.0/2.2/66.6	0.44	Mixed	2 - 7	0 - 9	PK-FK/FK-FK	1 - $2{\cdot}10^{10}$
VisiualGenome	Education	6	20	$3.6 \cdot 10^6$	6	73.4M	$3.1 \cdot 10^5$	-3.8/-0.4/2.0	0.20	Mixed	2 - 6	0 - 7	PK-FK/FK-FK	24 - $2{\cdot}10^8$

M3

R2D3

C DETAILED PROPERTIES OF DATASET

To cultivate a model capable of generalizing across diversified databases, we exhaustively search the public data repositories and obtain 30 datasets, including: 1) 6 well-established benchmark datasets for evaluating CardEst methods, such as TPC-H [10], SSB [61], IMDB [6] and STATS [13]; and 2) 24 new datasets assembled from [58]. We clean and process each dataset. We observe that these datasets closely resemble real-world scenarios and demonstrate notable diversity. Their detailed properties are listed in Table 4. We observe that these datasets closely resemble the real-world scenarios and demonstrate notable diversity, including:

1) *Domain*: These datasets include both synthetic and real-world data. For real-world data, it spans various domains such as government, sports, retail, and medicine, each differing significantly in scale, data distribution, and schema complexity (see details below).

2) *Scale*: The collection includes datasets in different volumes (from 2.9MB to 4.5GB); different numbers of tables (from 3 to 25), columns (from 12 to 359), and rows(from 1.29×10^4 to 6.85×10^7); and different domain size of attributes (from 7.1×10^3 to 1.44×10^7).

3) *Data Distribution*: The datasets exhibit significantly different levels of correlations among distributions of attributes. The average skewness scores of attribute distributions span from 0 to 93.6, and the average pairwise attribute RDC scores [52] (measuring the correlation) span from 0.04 to 0.53. This allows the pretrained model to learn different patterns to fit different joint PDFs.

4) *Join Scheme*: All datasets (except Airline and SSB) contain both primary-foreign key (PK-FK) and foreign-foreign key (FK-FK) joins. The structures of the join schema include the simple chain and star form of tables, as well as the complex forms in a mixture of chains and stars of tables.

D PSEUDO-CODE OF QUERY GENERATION

M3

R2D1

R2W3

Algorithm 1 Workload Generation for Datasets

```
Input: Datasets $D$, Number of queries to generate $N$
Output: Set of SQL queries for model training and evaluation
Construct the join schema graph $G_D$ with nodes (tables) $T_i$ and edges (join relationships) $T_i \sim T_j$
Enumerate all connected subgraphs $S \subseteq G_D$
for $i \leftarrow 1$ to $N$ do
    Randomly Select a subgraph $S_j$ from $S$
    Construct SQL query $Q$ using tables and join conditions from subgraph $S_j$
    $T \leftarrow$ Collect all nodes (tables) in subgraph $S_j$
    $A \leftarrow$ Collect all attributes associated with tables in $T$
    $m \leftarrow \mathrm{len}(A)$
    Sample an integer $n \sim \mathrm{Uniform}(1, m)$
    for $k \leftarrow 1$ to $n$ do
        Sample an attribute $A_i$ from the attribute set $A$
        if $A_i$ is a numerical attribute then
            Sample a lower bound $l \sim \mathrm{Uniform}(A_{i,\min}, A_{i,\max})$
            Sample an upper bound $u \sim \mathrm{Uniform}(l, A_{i,\max})$
            Define the filtering predicate as $l \le A_i \le u$
        else
            Sample a categorical value $v \sim \mathrm{Domain}(A_i)$
            Define the filtering predicate as $A_i = v$
        end if
        Append the filtering predicate to query $Q$
    end for
    Store the generated SQL query $Q$
end for
```

E COMPARISON BETWEEN ALECE AND PRICE

R1D4

Since both ALECE and PRICE utilize attention mechanisms and are applied in multi-table CardEst scenarios, we provide a detailed comparison to highlight the fundamental differences between the two models.

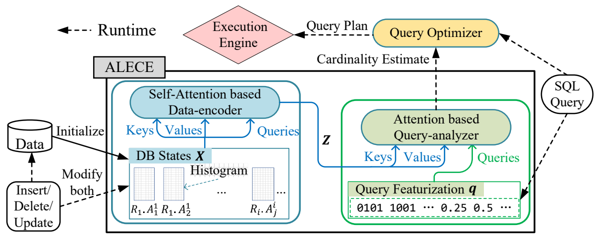


Figure 13: ALECE overview. (Extracted from [47])

ALECE Overview: On a high level, ALECE is designed to solve the CardEst problem of dynamic workloads on each specific database. ALECE employs a traditional transformer architecture for CardEst, as illustrated in Figure 13. During the encoder phase, ALECE applies histograms of all attributes in the dataset as individual tokens. In the decoder phase, these encoded histograms serve as keys and values, while the SQL statement, represented as a one-hot vector, acts as the query. This design essentially functions like a soft lookup table, linking each SQL query to the underlying data distribution to estimate the query's cardinality.

Because ALECE is tailored for each specific dataset, it relies on a hard-coded approach to map queries to underlying data. Specifically, it uses one-hot encoding to represent queries, and the model learns to map these hard-coded representations to the corresponding histograms during training. However, this encoding strategy is not transferable across databases. For example, a vector like $(0, 1, 0)$ might represent the "type" attribute in a "posts" table in one database, but this representation would be ineffective in another database where the "type" attribute does not exist or has a totally different distribution. Consequently, ALECE requires a separate model—essentially a dedicated lookup table—for each database to achieve accurate estimations.

PRICE Overview: In contrast, PRICE does not learn the specific mapping between SQL queries and data within a particular database. Instead, it aims at designing a pretrained cross-database model for CardEst, which is more broad and complex than ALECE. To this end, PRICE focuses on learning how to generalize from low-dimensional distributional features of attributes to the high-dimensional distributional features (i.e., cardinality) associated with SQL queries. From a statistical perspective, the cardinality of any SQL query can be derived by determining the probability of the range specified by its filtering predicates on the joint probability distribution functions (PDFs) of attributes across multiple tables. Therefore, the key tasks are selecting appropriate features, designing mechanisms to represent the joint PDFs, and calculating the probability based on filtering predicates.

To achieve this, PRICE employs transferable low-dimensional distributional statistics to represent the attributes in the database, rather than relying on non-transferable one-hot encoding. The attention mechanism then captures the necessary knowledge necessary for CardEst. PRICE simulates the fundamental process of obtaining cardinality by condensing high-level embeddings that represent the joint PDFs and applying filtering probabilities, making it generally applicable to any database.

Specifically, in PRICE, the features related to each join condition are first fused into embeddings that represent the backbone information of the table joining process. These embeddings are then combined with other attribute features to produce representations that reflect the details of the joint PDFs, allowing the model to filter probabilities effectively. The final cardinality is derived directly from these embeddings.

F EVALUATION ON ADAPTED JOB WORKLOAD

M2
M5
R2W2

The IMDB dataset encompasses two workloads: JOB and JOB-light. The JOB workload includes aggregation functions, LIKE queries, and OR conditions, which are not supported by most existing ML-based cardinality estimation (CardEst) methods, including our model, PRICE. Our focus has primarily been on Select-Projection-Join (SPJ) queries, which are more commonly addressed in the literature. Consequently, we initially used JOB-light for our evaluations, as it aligns with recent CardEst research practices[29, 39, 47, 60, 87].

Table 5: Evaluation on JOB-Adapt workload.

CARDSET	E2E	Q-ERROR				P-ERROR			
METHOD	TIME (S)	50%	90%	95%	99%	50%	90%	95%	99%
PG	9623	2.88	25.48	64.27	678.87	1.07	2.06	3.71	9.08
PRICE	7735	2.43	14.96	35.75	334.74	1.02	1.25	1.70	5.78
Optimal	7280	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

However, to assess PRICE's performance on more complex queries, we adapted the JOB workload to create JOB-Adapt. Unlike JOB-light, JOB includes intricate multi-table join queries, involving up to 17 tables. We preserve the join conditions from the JOB templates and generate supported filtering predicates that are compatible with PRICE. Specifically, we generate 99 queries based on JOB's 33 templates, which are associated with 13,247 sub-queries. As demonstrated in Table 5, PRICE performs effectively on this adapted JOB workload, outperforming PostgreSQL (PG) and approaching near-optimal end-to-end performance. This evaluation result aligns with our reported performance of PRICE on other datasets in the manuscript and further demonstrates PRICE's robustness in handling more complex query scenarios.