Proceedings of Machine Learning Research vol 245:1–13, 2024 2024 International Conference on Machine Learning and Intelligent Computing

A Tree-Structure Enhanced Transformer for Cardinality Estimation

Mingjie Hu HMJ_9906@NUDT.EDU.CN
Department of Intelligent Data Science, National University of Defense Technology

Qihang Zhang 17608496514@163.COM
Department of Intelligent Data Science, National University of Defense Technology

Jing Ren[*] RENJING@NUDT.EDU.CN
Department of Intelligent Data Science, National University of Defense Technology

Hengzhu Liu HENGZHU_LIU@263.NET
College of Computer, National University of Defense Technology

Editors: Nianyin Zeng and Ram Bilas Pachori

Abstract

Accurate cardinality estimation is crucial for query optimization by guiding plan selection. Traditional cardinality estimation approaches often fail to provide precise estimates, leading to suboptimal query plans. In recent years, learning-based methods have emerged as a promising alternative. For tree-based learning methods, a conventional way to simply encode nearby parent-child pairs and learn by iteratively training hampers the performance. But it leads to a significant loss of structural information and incurs substantial computational overhead by the iterative training process. In this paper, we proposed a tree-structure positional encoding scheme. It can not only extract effective features for each node, but also capture the inherent structural characteristics of the tree. Based on the tree-based feature, we designed a novel transformer-based cardinality estimation model, which enhances the parallelism of the model training process and reduces the overhead caused by iterative training. On real-world datasets, our method beats the current state-of-the-art techniques, QF, by 25% in terms of mean qerror.

Keywords: Cardinality estimation, Tree Transformer, Positional Encoding

1. Introduction

Cardinality estimation plays a pivotal role in optimizing database queries. Modern database query optimizers heavily rely on their internal cost estimation systems, where the cardinality of a query operator serves as a critical factor. Cardinality refers to the number of result rows that remain after filtering data through the query conditions within an operator. Accurate and efficient cardinality estimation is essential to optimize the execution plan and enhance the overall performance of the optimizer. It is often regarded as the ”Achilles heel” of query optimizers (Lohman, 2014).

Cardinality estimation methods have been extensively studied in academia and industry for decades. However, studies (Leis et al., 2015, 2018) indicate that traditional query optimizers based on cost models often select sub-optimal or even inferior execution plans. This is primarily due to the trade-off they have to make between marginal improvement in cardinality estimation accuracy and incurring significant performance overhead.

Recent researches (Qiao et al., 2021; Sun and Li, 2019; Zhao et al., 2022; Zhu et al., 2023; Sun et al., 2022; Woltmann et al., 2023; Han et al., 2022; Chen et al., 2023; Mikhaylov et al., 2022) indicate that learning-based methods provide a more effective way to perform cardinality estimation. Fig.1 provides a unified workflow for a learning-based estimator. The query plan represents the logical and physical operations that the system performs to retrieve data from the database. Each node in the query plan tree is first encoded with various statistics from the database, for example, Sample [7]. Subsequently, the encoded vector tree undergoes neural network processing, and the resulting output is then fed back into the database for further processing.

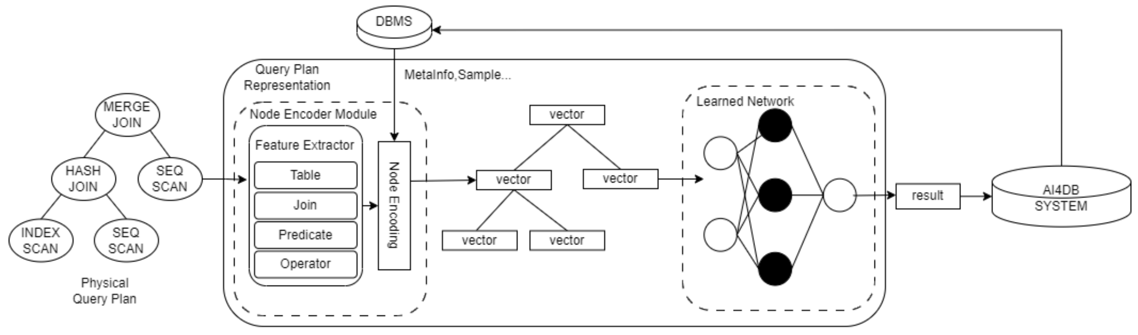


Figure 1: A unified workflow of learning-based cardinality estimator.

Tree-CNN (Mou et al., 2016) is a specialized form of traditional CNN that allows for input to tree structures. BAO (Marcus et al., 2021) employ Tree-CNN along with triangular-shape filters to slide over the query plan tree. This approach enables the capture of parent-child dependencies within the query plan tree. Tree-CNN is easily trained. However, this method has a small receptive domain, and each node can only "see" the characteristics of neighboring nodes. As a result, it cannot capture long information flow paths from leaf nodes to root nodes. E2E (Sun and Li, 2019) is one of the most advanced methods available for estimating cardinality and cost. It utilizes the Tree-LSTM (Tai et al., 2015) model to overcome gradient vanishing and explosion issues caused by complex queries. MSP (Liu et al., 2022) developed two network models, DTree-LSTM and CTree-LSTM, and incorporated MetaInfo. But LSTM models can be challenging and time-consuming to train due to their recursive steps. Additionally, recursive processing in query plan trees with long information flow paths can lead to information loss from leaf nodes before it reaches the top node. Transformer has excellent performance in various fields and is one of the current research hotspots (Zhang et al., 2023; Yan et al., 2022). QueryFormer (Zhao et al., 2022) first uses Transformer architecture in query plan representation. And as far as we know, it is the state-of-the-art method currently. However, QueryFormer does not deeply consider the structural information of tree data and the role of positional encoding in Transformer. It directly add the height embedding with the node encoding, and forced tree-bias is applied to set the attention matrix.

The primary distinctions among these studies are the distinct node feature extraction methods and the specific network model they employ. However, a common limitation of these methods of these approaches is their insufficient ability to model long information flow paths and effectively capture positional information. This stems from their failure to consider the structural characteristics of the query plan tree when constructing its representation.

The representation of a query plan is of great importance in learning-based cardinality estimation, as it enables the identification of potential performance issues and facilitates query optimization. In this paper, we introduce a refined tree positional encoding scheme, which adeptly captures both the node-specific attributes and the aggregate characteristics of the query plan. Moreover, we propose a novel model called Tree-Transformer model for Cardinality Estimation (TTCE), to incorporate the above schema to transformer. Through experiments, we demonstrate that our proposed method significantly improves the performance. Our contributions are as follows:

- We design an efficient tree positional encoding schema, that incorporates the path encoding and the relationship encoding of the query plan tree. It can effectively describe the structural information of the tree.
- We propose TTCE, a Tree-Transformer model for Cardinality Estimation, which can effectively characterize the query plan tree, including its ability to extract long-path information flow and effectively combine the structural information of each node with the node characteristics.
- We conduct experiments on real-world datasets. The results indicate that our approach outperforms previous state-of-the-art methods.

2. Tree-Based Encoding Schema

In this section, we present the design of tree-based encoding schema, including a **node encoding schema** which extracts effective features of each node in the query plan tree, along with the proposed **tree positional encoding schema** that captures the inherent structural characteristics of the tree. The tree-based encoding schema is the foundation for representing the query plan tree.

2.1. Node Encoding Schema

Node encoding schema typically obtain necessary information from node characteristics and database statistics, covering elements such as *operation*, *predicate*, *join*, and *table* in query plan trees. *Operation* describes the physical operation of query plan nodes and is a categorical variable with finite fields (e.g., about 30 in PostgreSQL (PostgreSQL, 2023)). *Predicate*, defining filter conditions for relational table columns, can be represented as triples in the form of <*column, operator, value*>, with *column and operator* treated as categorical variables, and value normalized to the range of (0,1). *Join*, representing a connection condition between tables, and *table*, referring to the relational table associated with node operations, are also considered categorical variables.

Existing methods often use ”one-hot” encoding for *operation*, *table*, and *join* (Kipf et al., 2018; Liu et al., 2022; Marcus and Papaemmanouil), but this method has insurmountable limitations. A single table in a large database may have hundreds of columns. It thus requires hundreds of bits for column encoding alone. When the database is updated and new categorical variables are introduced, it would be unavoidable to retrain the entire machine learning system. To overcome these limitations, we use fixed-size embedding vectors (Zhao et al., 2022) to represent each categorical variable, allowing for efficient updates and the introduction of new categories without disrupting existing embeddings.

For *Operation*, *Join*, and *Table*, their vector representations can be denoted as E_o, E_j and E_t, respectively. *Predicates* are characterized as triples <*column, operator, value*>. We concatenate the embeddings of each element in the triplet as previous work (Sun and Li, 2019; Zhao et al., 2022; Kipf et al., 2018), and the we obtain the predicate representation, denoted as $E_p = [E_c, E_o, v_{norm}]$. E_c represents the embedding vector of the column, E_o represents the embedding vector of the operator, and v_{norm} represents the normalized value of the predicate.

Statistics contain the distribution information of the data in the database table, and there are a variety of statistics to choose from, such as histogram (PostgreSQL, 2023), MetaInfo (Liu et al., 2022), etc. We choose the widely used statistics, Sample (Kipf et al., 2018). We randomly sample from each table, and then we get a set of m tuples. For each predicate in the query node, we get an m-bit bitmap by assessing whether each extracted tuple satisfies the predicate condition. Following the approach employed in previous studies (Sun and Li, 2019; Zhao et al., 2022; Kipf et al., 2018), we set m=1000. We denote this sample bitmap as E_s.

By employing the aforementioned method, we obtain five vectors: E_o, E_j, E_t, E_p, and E_s. These vectors are separately passed through corresponding linear layers, and the resulting embeddings are concatenated to obtain the final embedding of the node as *E=[Linearo(E_o), Linearj(E_j), Lineart(E_t), Linearp(E_p), Linears(E_s)].*

2.2. Tree Positional Encoding Schema

The Transformer model typically operates on sequential input (Su et al., 2024). Therefore, when dealing with tree-structured data, it becomes necessary to serializer the data before passing it to the Transformer. Various methods have been proposed for serialization, including Depth-First Search (DFS) and Breadth-First Search (BFS). In our approach, we employ BFS (Breadth-First Search) to serialize the query plan tree.

Figure 2 illustrates the BFS sequence order for a specific query plan tree. However, as depicted in the figure, the serialization process leads to the loss of structural information inherent in the tree. Structural information holds great importance for a query plan tree, and losing such information is unacceptable.

To capture this information, we can incorporate features like the location and parent-child relationship of each node in the query plan tree.These encoding methods, termed path encoding and relationship encoding, are similar to the absolute positional encoding and relative positional encoding used in Transformer model (Lin et al., 2022; Dufter et al., 2022). Consequently, the tree's structural information is effectively incorporated.

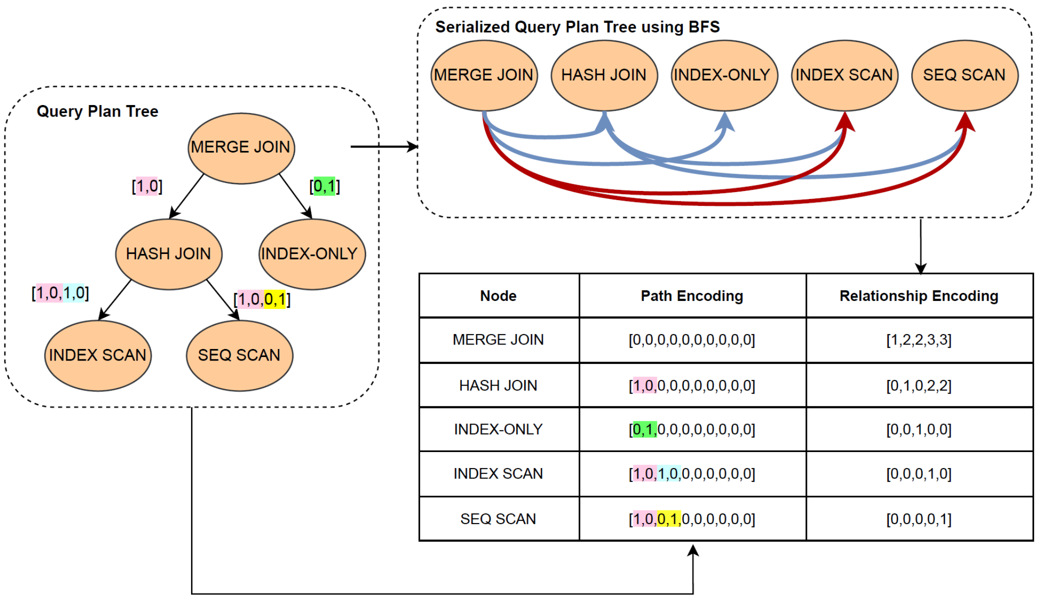
2.2.1. PATH ENCODING

A previous study (Shiv and Quirk, 2019) introduced a novel positional encoding method that represents the path from the root node to the target node by concatenating the path information of each node in reverse order. However, with this method, nodes that share similar paths exhibit discrete encoding results in vector space. In our modification, we enhance this concatenation method by employing forward concatenation along with padding. This approach effectively preserves the absolute position information for each node.

As showed in Figure 2, the length of the encoding is twice the number of nodes and paths are depicted as numbers on a red background. In the given example, there are five nodes, resulting in a encoding length of 10. Each path selection can be represented by two digits [0,1] or [1,0], indicating the left path or right path, respectively. The use of two bits ensures that padding does not introduce ambiguity. For MERGE JOIN node, which is the root node, there is no path from the root node to itself (i.e., the path is empty), Padding is used to increase the length of the vector to 10. In this case, the path is encoded as an all-zero vector. For SEQ SCAN node, the path from the root node to the target node is represented as [0,1] for the left path, followed by [1,0] for the right path. After concatenation and padding, the path is encoded as [0,1,1,0,0,0,0,0,0,0,0,0].

To obtain the path encoding corresponding to each node (denoted as $P_p^{'}$), it is passed through a linear layer, which maps $P_p^{'}$ to a higher-dimensional feature space. The resulting path encoding is denoted as P_p.

$$P_p = ReLU(W_p P_p^{'} + b_p) \tag{1}$$



Node	Path Encoding	Relationship Encoding
MERGE JOIN	[0,0,0,0,0,0,0,0,0,0]	[1,2,2,3,3]
HASH JOIN	[1,0,0,0,0,0,0,0,0,0]	[0,1,0,2,2]
INDEX-ONLY	[0,1,0,0,0,0,0,0,0,0]	[0,0,1,0,0]
INDEX SCAN	[1,0,1,0,0,0,0,0,0,0]	[0,0,0,1,0]
SEQ SCAN	[1,0,0,1,0,0,0,0,0,0]	[0,0,0,0,1]

Figure 2: A example of BFS serialization and tree positional encoding schema.

2.2.2. RELATIONSHIP ENCODING

To encode the relationship of nodes in a tree, we use the shortest path algorithm (e.g., Dijkstra, Floyd). Since each node in the query plan tree only depends on its child nodes, we only need to consider the distance between a node and its children. For other nodes that are unreachable from a node, we assign zero as the distance. For instance, for the MERGE JOIN node in Figure 2, all nodes are reachable. We set the distance to itself as one, the corresponding encoding result is [1,2,2,3,3], where each element corresponds to a serialized node. We denote the relationship encoding for each node as $P_r^{'}$. We apply the same operation on $P_r^{'}$ as on $P_p^{'}$ to obtain P_r.

$$P_r = ReLU(W_r P_r^{'} + b_r) \tag{2}$$

3. The Architecture of TTCE Model

In this section, we introduce the architecture of our TTCE. We first describe the Transformer architecture that we used, and then introduce our methodology for utilizing tree-structured information with the idea of TUPE (Transformer with United Positional Encoding) (Ke et al., 2022). Figure 3 illustrates the architecture of the TTCE.

3.1. Representation Layer

The Transformer architecture can be applied in three different ways (Lin et al., 2022), Encode-Decoder, Encoder-only, and Decoder-only. The cardinality estimation task involves extracting relevant features from a query plan tree and generating an estimated cardinality. The Encoder-only architecture is well-suited for this task. We have made minimal modifications to the original Transformer architecture and have retained the same components, such as the multi-head attention mechanism and feed-forward network (FFN).

TUPE (Ke et al., 2022) introduces a idea that untying the correlations between words and their respective positions. We are inspired by the idea of TUPE. The calculation of each component in Affinity Matrix is expressed by Eq.3, where w represents the input sequence data and p represents the positional encoding. Positional encodings were combined with word embeddings as input to Transformer. The two types of information are inherently distinct. Positional encoding represents the index of word, which is not semantic and very different from the meaning of words (Shiv and Quirk, 2019; Ke et al., 2022).

$$a_{ij} = \frac{1}{\sqrt{d_k}}((w_i + p_i)W^{Q,l})((w_j + p_j)W^{K,l})^T = \\ \frac{(w_i W^{Q,l})(w_j W^{K,l})^T}{\sqrt{d_k}} + \frac{(w_i W^{Q,l})(p_j W^{K,l})^T}{\sqrt{d_k}} + \frac{(p_i W^{Q,l})(w_j W^{K,l})^T}{\sqrt{d_k}} + \frac{(p_i W^{Q,l})(p_j W^{K,l})^T}{\sqrt{d_k}} \tag{3}$$

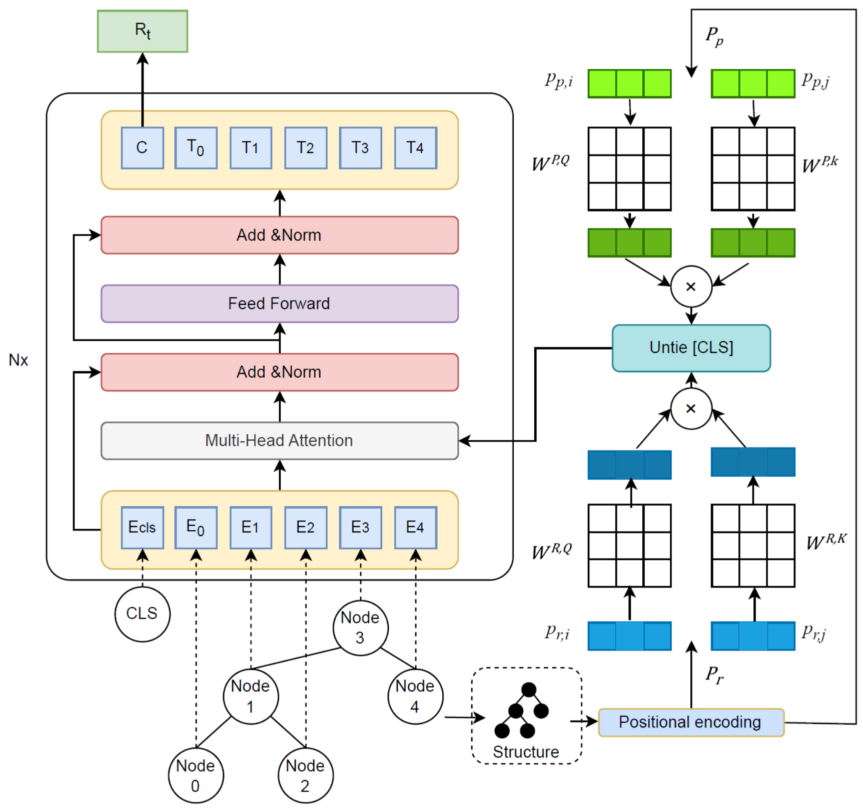


Figure 3: The overall structure of Tree-Transformer model for Cardinality Estimation(TTCE).

Similarly, the processed information of positional embedding and node embedding has very different meanings. It is more sensible to utilize separate matrices for these calculations. Consequently, in our TTCE, Eq.3 can be formulated as Eq.4 and Eq.5. And the calculation of Affinity Matrix can be rewritten as Eq.6. In our design, the matrix P is computed once and then added to the attention calculation in each encoder module. The additional time overhead incurred by this operation is minimal.

$$a_{ij} = \frac{1}{\sqrt{d_k}}((w_i)W^{Q,l})((w_j)W^{K,l})^T + p_{ij} \tag{4}$$

$$p_{ij} = \frac{1}{\sqrt{d_k}}((p_i^{Pat})W_{Pat}^{Q})((p_j^{Pat})W_{Pat}^{K})^T + \frac{1}{\sqrt{d_k}}((p_i^{Rel})W_{Rel}^{Q})((p_j^{Rel})W_{Rel}^{K})^T \tag{5}$$

$$Attention(Q,K,V,P) = softmax(\frac{QK^T + P}{\sqrt{d_k}}) \tag{6}$$

3.2. [CLS] Node

Our model uses [CLS] node (Devlin et al., 2018) to enhance the performance of TTCE. We connect the [CLS] node to all nodes in the query plan by resetting the relevant values in P. The specific reset method is shown in Eq.7, where $\theta = \{\theta_1, \theta_2\}$ is a learnable parameter. This operation is called "Untie [CLS]" in Figure 3.

"Untie [CLS]" makes the [CLS] node attend all nodes in the query plan tree and potentially emphasize important nodes. Moreover, it can distinguish the [CLS] node from other nodes and acknowledge its unique role in the model. During training, the [CLS] node is updated alongside other tree nodes. The output vector of the [CLS] node represents the entire input tree as a fixed-size vector that can be used for downstream task.

$$reset(p,i,j) = \begin{cases} p_{ij}, i \neq 0, j \neq 0, (no\ related\ to\ [CLS]) \\ \frac{\theta_1+\theta_2}{2}, i = j = 0, ([CLS]\ node) \\ \theta_1, (from\ [CLS]\ to\ other\ nodes) \\ \theta_2, (from\ other\ nodes\ to\ [CLS]) \end{cases} \tag{7}$$

3.3. Prediction Layer

The prediction layer of our model is a two-layer fully connected neural network, the first layer applies the *ReLU* activation function, and the second layer uses the *Sigmoid* function to predict the cardinality. The input to the output layer is obtained from the encode module's output. The output layer can predict the cardinality of any plan from the representation vector of the plan. The calculation formula is as follows:

$$\begin{gathered} card^{'} = ReLU(W_{card'}R_t + b_{card'}) \\ card = Sigmoid(W_{card}card^{'} + b_{card}) \end{gathered} \tag{8}$$

Card represents the result of cardinality normalization of the output. The final *Sigmoid* output value ranges between 0 and 1, and it is further transformed into cardinality using the inverse of the normalization formula. In the last layer of the prediction layer, we utilize the *Sigmoid* activation function. This is because the value range of the *Sigmoid* function is (0,1), which aligns with the value range of the formula Eq.9. We choose logarithmic normalization as the normalization method, and we set $log(card_{min})$ and $log(card_{max})$ to 0 and 50 , respectively.

$$card\_norm_i = \frac{log(card_i) - log(card_{min})}{log(card_{max}) - log(card_{min})} \tag{9}$$

4. Experiment

4.1. Dataset

We utilized the IMDB (Leis et al., 2015). The IMDB dataset contains 21 tables covering more than 2.5 million films produced by 234,997 company over 133 years, involving more than 4 million actors. We use 100k queries on this dataset, and these queries contain only 0-2 joins. The queries are divided into a training set and a validation set, with a ratio of 9:1. The training set and division followed the same approach as in previous work (Zhao et al., 2022; Kipf et al., 2018).

Table 1: Two types of workloads.

Workload	Joins	Predicate
Synthetic (Kipf et al., 2018)	0-2	=, >, <
JOB-light (Kipf et al., 2018)	1-4	=, >, <
IMDB-NUMS	0-5	=, >, <, !=, <=, >=

4.2. Experimental Setting

As shown in Table 1, we employ two types of query workloads in our study. As we have not considered string-based predicates in this context, the workloads exclusively consist of queries involving numerical operations. By limiting the scope to numeric predicates, we can effectively analyze and evaluate the performance of our approach within this specific domain.

The first workload includes Synthesis and JOB-light (Kipf et al., 2018) workloads. The Synthesis workload consists of 5000 queries with up to two joins. The JOB-light workload consists of 1-4 joins, for a total of 70 queries. Synthetic workload is generated using the same script and random number seed as the training data. The JOB-light workload is derived from the Join Order Benchmark (JOB) (Leis et al., 2015). The second workload is used to test the adaptability of the model. This workload contains predicates that were not used in the original training set, such as !=, >=, and the number of joins is different from the training data. We refer to this workload as IMDB-NUMS for easy explanation.

We use qerror as the evaluation metric. The calculation formula is as Eq.10, where $card_i$ is the actual cardinality and est_card_i is the estimated cardinality.

$$qerror(card_i, est\_card_i) = \frac{max(card_i, est\_card_i)}{min(card_i, est\_card_i)} \tag{10}$$

We experimented with the Intel(R) Core(TM) CPU i7-12700F, 64GB of RAM, and the GeForce RTX 3060. For the Transformer backbone, we set the number of headers to 8, the layers to 6, and dropout to 0.1. We trained with the Adam optimizer with a learning rate of 0.001. Our training method took approximately 30s to complete a single epoch, while QF (Zhao et al., 2022) took approximately 40s in our machine.

4.3. Experimental Results

We employ different methods for training on training set. After the model converges, test with the first workload to get the result. Figure 4 shows how mean-qerror changes on the validation set during training. The model tends to converge after 30 epochs, and the results on the validation set indicate that the model with two positional encodings experiences a lower convergence value.

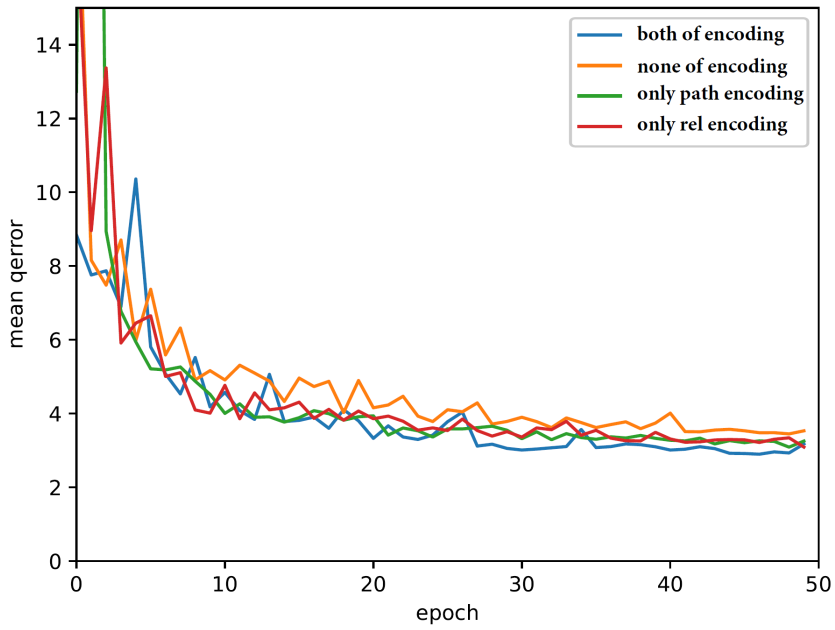


Figure 4: Mean-qerror changes on the validation set during training.

Table 2 shows the test results for different positional encodings. Mean is the average error of all queries tested. Median is the median number of errors across all queries tested. The 90th means the 90th percentile of errors across all queries tested. We also report results for PostgreSQL (PostgreSQL, 2023), MSCN (Kipf et al., 2018), QueryFormer (Zhao et al., 2022) for reference.

Table 2: Q-error on test workloads.

	Synthetic			JOB-light		
	Mean	Median	90th	Mean	Median	90th
Postgresql (Ke et al., 2022)	25.44	2.10	13.56	162.34	7.95	157.70
MSCN (Kipf et al., 2018)	2.89	1.18	3.32	57.90	3.82	78.40
QueryFormer (Zhao et al., 2022)	2.72	**1.11**	3.34	29.50	2.26	38.74
E2E (Sun and Li, 2019)	5.30	2.19	7.97	56.85	2.43	39.29
TTCE(no positional encoding)	2.42	1.21	3.18	37.70	2.57	35.11
TTCE(only relationship encoding)	2.28	1.15	**2.99**	45.27	2.56	54.32
TTCE(only path encoding)	2.31	1.38	3.05	32.79	2.55	**33.27**
TTCE(both of encoding)	**2.26**	1.23	3.15	**15.86**	**2.04**	38.58

Effectiveness of TTCE for cardinality estimation. As shown in Table 2, TTCE has a significant improvement over Q-error compared with other methods. This shows that it can better capture the relevant characteristics of the query plan tree, thereby producing more accurate results. For the most important evaluation indicator, mean q-error, TTCE containing two types of positional encoding are 25% better for synthetic workloads and 50% better for job-light workloads than the previous best method QF. The effectiveness of our approach is fully proven.

Ablation Analysis. We conducted experiments to evaluate two types of positional encoding designs: using only path encoding, using only relationship encoding, and not using positional encoding, as well as using both positional encodings. As show in Table 2, the combination of the two positional encodings outperformed the scenarios where positional encodings were not used or only one type of positional encoding was employed, across most metrics. This shows that the encoding schema we designed can effectively improve the prediction accuracy.

4.4. Fine-Tune

A significant limitation of learning-based cardinality estimators is their tendency to perform poorly on workloads that deviate significantly from the original training workloads. When workloads change, learning-based estimator must be either re-trained or rendered unusable for the new scenario. But retraining requires significant time.

Transformer can be adapted to various scenarios by pre-training + fine-tuning (Yu et al., 2022). We tested this property of TTCE. Taking the previously trained model as the pre-training model, we fine-tune the model with 1k queries different from the training data as new training data, using the Adam optimizer with a learning rate of 0.0001, and train 10 epochs, which takes about 5s. We test on 100k queries. As shown in Table 3, we can see TTCE performs quite poorly before Fine-tune for new workloads, and Fine-tuning has significantly improved the accuracy of TTCE for IMDB-NUMS workload with little overhead. This experiment demonstrates the effectiveness of TTCE in handling new workloads with minimal overhead by fine-tuning.

Table 3: Fine-tune performance on IMDB-NUMS workload.

	IMDB-NUMS		
	Mean	Median	90th
Before Fine-tune	22560.70	2.04	4959.21
After Fine-tune	52.96	2.36	16.85

5. Conclusion

In this paper, we introduce TTCE, a Tree-Transformer model for cardinality estimation. We propose a novel schema using path encoding and relationship encoding to preserve the structural characteristics of query plan trees. This approach significantly enhances the accuracy of cardinality estimation. Our experiments on real datasets demonstrate that our method outperforms existing techniques.

Our research offers substantial benefits for Database Management Systems by providing more precise cardinality estimations, which can markedly improve query performance. While TTCE can quickly adapt to changing workloads through fine-tuning, it's not ideal to do it every time workload drift occurs. This would require a very expensive data collection process. We will further focus on developing a more robust method for cardinality estimation that efficiently handles changing workloads.

Acknowledgments

This work was supported by The National Natural Science Foundation of China:62101575 and The Self-directed Project of State Key Laboratory of High Performance Computing:202101-18.

References

Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. Loger: A learned optimizer towards generating efficient and robust query execution plans. *Proceedings of the VLDB Endowment*, 16(7):1777–1789, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pages 4171—-4186. Association for Computational Linguistics, Minneapolis, Minnesota, 2018.

Philipp Dufter, Martin Schmitt, and Hinrich Schütze. Position information in transformers: An overview. *Computational Linguistics*, 48(3):733–763, 2022.

Yi Han, Linbo Qiao, Dongsheng Li, and Xiangke Liao. Review of knowledge-enhanced pre-trained language models. *Journal of Frontiers of Computer Science & Technology*, 16(7), 2022.

Guolin Ke, Di He, and Tie-Yan Liu. Rethinking positional encoding in language pre-training. international conference on learning representations, 2022.

Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.

Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3): 204–215, 2015.

Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *The VLDB Journal*, 27:643–668, 2018.

Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *AI open*, 3: 111–132, 2022.

Honghao Liu, Zhiyong Peng, Zhe Zhang, Huan Jiang, and Yuwei Peng. Msp: Learned query performance prediction using metainfo and structure of plans. In *Web and Big Data: 6th International Joint Conference, APWeb-WAIM 2022, Nanjing, China, November 25–27, 2022, Proceedings, Part III*, pages 3–18. Springer, 2022.

Guy Lohman. Is query optimization a "solved" problem. In *Proc. Workshop on Database Query Optimization, Vol. 13. Oregon Graduate Center Comp. Sci. Tech. Rep*, page 10, 2014.

Ryan Marcus and Olga Papaemmanouil. Plan-structured deep neural network models for query performance prediction. In *Proc. VLDB Endow*, volume 12.

Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1275–1288, 2021.

Artem Mikhaylov, Nina S Mazyavkina, Mikhail Salnikov, Ilya Trofimov, Fu Qiang, and Evgeny Burnaev. Learned query optimizers: Evaluation and improvement. *IEEE Access*, 10:75205–75218, 2022.

Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*, volume 30, pages 1287–1293. AAAI Press, 2016.

PostgreSQL. Postgresql 12.15 documentation, June 2023. URL `https://www.postgresql.org/docs/12/`.

Shao-Jie Qiao, Guo-Ping Yang, Nan Han, Hao Chen, Fa-Liang Huang, Kun Yue, Yu-Gen Yi, and Chang-An Yuan. Cardinality estimator: processing sql with a vertical scanning convolutional neural network. *Journal of Computer Science and Technology*, 36(4):762–777, 2021.

Vighnesh Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 1082–12091. Curran Associates Inc., Red Hook, NY, USA, 2019.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ji Sun and Guoliang Li. An end-to-end learning-based cost estimator. *Proceedings of the VLDB Endowment*, 13(3):307–319, 2019.

Luming Sun, Tao Ji, Cuiping Li, and Hong Chen. Deepo: A learned query optimizer. In *Proceedings of the 2022 International Conference on Management of Data (Philadelphia, PA, USA) (SIGMOD '22)*, pages 2421–2424. Association for Computing Machinery, New York, NY, USA, 2022.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. pages 1556–1566, 2015.

Lucas Woltmann, Jerome Thiessat, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. Fastgres: Making learned query optimizer hinting effective. *Proceedings of the VLDB Endowment*, 16(11):3310–3322, 2023.

Bo Yan, Siwei Wang, En Zhu, Xinwang Liu, and Wei Chen. Group-attention transformer for fine-grained image recognition. In *Advances in Artificial Intelligence and Security: 8th International Conference on Artificial Intelligence and Security, ICAIS 2022, Qinghai, China, July 15–20, 2022, Proceedings, Part II*, pages 40–54. Springer, 2022.

Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. Cost-based or learning-based? a hybrid query optimizer for query plan selection. *Proceedings of the VLDB Endowment*, 15(13):3924–3936, 2022.

Yuxuan Zhang, Huibin Tan, Long Lan, Jing Ren, and Xiao Teng. Person re-identification based on multi-spectrum information aggregation transformers. In *2023 8th International Conference on Image, Vision and Computing (ICIVC)*, pages 131–135. IEEE, 2023.

Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. Queryformer: A tree transformer model for query plan representation. *Proceedings of the VLDB Endowment*, 15(8):1658–1670, 2022.

Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. Lero: A learning-to-rank query optimizer. *Proceedings of the VLDB Endowment*, 16(6):1466–1479, 2023.