MoEPlan: A Lazy Learned Query-Selection Optimizer via Mixture of Optimizer Experts

Suchen Liu[1], Jun Gao[1](✉), Yinjun Han[2](✉), and Yang Lin[2]

[1] Key Laboratory of High Confidence Software Technologies, EECS, Peking University, China
{2301111946,gaojun}@pku.edu.cn
[2] ZTE Corporation
{han.yinjun,lin.yang}@zte.com.cn

Abstract. The learned plan-selection optimizers, which generate diverse candidate plans via different traditional optimizers and then select the best expected performance plan using a learned value model, can achieve stable and relatively efficient performance by combining the advantages of conventional and learned methods. However, these eagerly-generated plans incur high optimization overhead, as they require multiple invocations of the native optimizer.
In this paper, we propose MoEPlan, a method that learns a routing policy to select the top-k experts (different optimizers) by directly matching query embedding and expert embeddings, without needing to explicitly obtain execution plans from each expert in advance. MoEPlan also incorporates a virtual ideal expert, which is trained independently and guides the selection of the best plan of the top-k experts through learned plan similarities. Experimental studies demonstrate that MoEPlan, with only two plans generated, takes less inference time, while still producing more efficient plans than other learned plan-selection optimizers.

Keywords: DB · Query Optimizer · Lazy · Mixture-of-Expert

1 Introduction

Database optimizers in DBMS generate efficient execution plans by enumerating candidate sub-plans and selecting the best one based on cost estimation. Finding an optimal join order is NP-complete [4]. While native DBMS optimizers achieve high performance for diverse queries, they still struggle with complex data distributions [7].

To address these challenges, researchers have applied deep learning to query optimizers [8, 17, 7, 16, 1, 19, 15]. Existing learned optimizers can be roughly divided into two categories: *learned plan-generation optimizers* and *learned plan-selection optimizers*. Learned plan-generation optimizers use reinforcement learning for plan construction but face high training costs and unstable performance [7]. In contrast, learned plan-selection optimizers, such as Bao [7] and Lero [19], leverage existing native optimizers to generate multiple execution plans and select the most efficient one using tree-structured models (*e.g.,* tree-LSTM [12], tree-CNN [10]).

However, as noted by Bao [7], learned plan-selection methods require significant optimization time, hindering their effectiveness. The time cost involves three parts: obtaining multiple execution plans with different hints, encoding the plans, and using

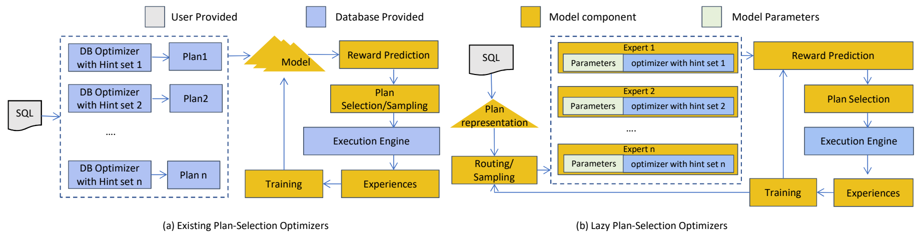


Fig. 1: Method Differences

either absolute pointwise [7] or relative pairwise [19] models for final plan selection. Since the optimizer must be invoked multiple times, the optimization time for these methods is substantially higher than that for the native DBMS optimizer.

The key limitation of existing methods is their inability to capture query features initially, requiring queries to be submitted to multiple optimizers. To address this, we propose *MoEPlan*, a learned plan-selection optimizer shown in Figure 2 that routes queries to a subset of promising experts without generating execution plans in advance. By matching query and expert embeddings to predict execution times, MoEPlan significantly reduces optimization overhead. The contributions of this paper are as follows:

– We introduce MoEPlan, a learned plan-selection optimizer comprising query representation, routing, and expert components. We utilize QueryFormer [18] to embed queries and a routing network to direct queries to promising experts. These experts estimate execution time from the query embedding, and the best-performing one generates and evaluates the plan. Such a lazy method can significantly lower the time cost in the inference phase.
– We enhance MoEPlan by incorporating a virtual ideal expert in the expert component to guide plan selection based on plan similarity.
– Experiments across datasets demonstrate that MoEPlan outperforms existing methods in plan generation efficiency and plan effectiveness.

The remainder of this paper is organized as follows. We review the background knowledge and problem formulation in Section 2. Then, we describe the MoEPlan in Section 3. Section 4 reports experimental results. We review related works in Section 5 and conclude this paper in Section 6.

2 Background and Problem Formulation

This section reviews background knowledge, including database hints and the gating network in MoE, followed by the formulation of the problem studied in this paper.

2.1 Database Hint

Database hints enable users to modify the DBMS optimizer's behavior and search space non-intrusively [13], including specifying physical operators or adjusting cardinalities.

In learned plan-selection methods, Bao [7] uses coarse operator specifications, while Lero [19] focuses on cardinality. Hints help generate diverse plans for learning data distributions and improving cardinality estimation [7]. We adopt Bao's hint setting for its simplicity and ability to generate diverse query plans.

2.2 Gating Network in MoE

MoE uses multiple moderate complexity models (experts) instead of a single large model to fit the entire dataset [5]. The gating network assigns weights to each expert, encouraging competition or cooperation through different loss functions.

MoE has surged in popularity with large models [3], with a gating network selecting top-k experts for scalability. To avoid skewed training, random noise is added before selection, and the loss function promotes balanced training [3]. The Switch Transformer [3] selects one expert at a time, supporting more experts with more parameters.

MoE's theoretical basis, though unclear, is supported by evidence that input clustering and expert non-linearity contribute to performance [2]. We apply MoE to our optimizer, as different optimizers excel with similar query features, forming necessary clusters. The gating network in MoEPlan identifies query-expert patterns, routing queries to the best experts, similar to the Switch Transformer.

2.3 Problem Formulation

We define the symbols used as follows: Let O be a optimizer with optimizer experts E, and Q be a query set. The time consumed by Q using O includes expert selection time $T_s(O, Q)$ and plan execution time $T_e(O, Q) = \sum_{q \in Q} t_e(e_O(q), q)$, where the former is the total time to select the best expert $e_O(q)$ for each $q \in Q$, and the latter is the sum of execution time for the selected expert $e_O(q)$ executing each query q. Following Bao [7], the regret of O is defined as $Reg(O, Q) = \sum_{q \in Q} t_e(e_O(q), q) - t_e(e_q^*, q)$, measuring the cumulative difference between the execution time of the plan generated by the expert $e_O(q)$ and the optimal expert e_q^* for q. Our goal is to develop a learned plan-selection optimizer O that minimizes both the expert selection time $T_s(O, Q)$ and the regret $Reg(O, Q)$.

3 MoEPlan for Plan Selection

In this section, we first outline the framework of MoEPlan, detail the components of MoEPlan, and finally provide an efficiency analysis.

3.1 Architecture

MoEPlan includes three key components: query representation, routing, and experts. As shown in Figure 2, both training and inference stages follow a similar flow. The query q is converted into an embedding by the query representation component. The routing component selects top-k experts and samples additional ones to avoid missing optimal experts. Each expert has an embedding, a prediction network, and a built-in optimizer. During training, selected experts generate ground-truth execution times T; during inference, the plan with the minimum predicted time is chosen.

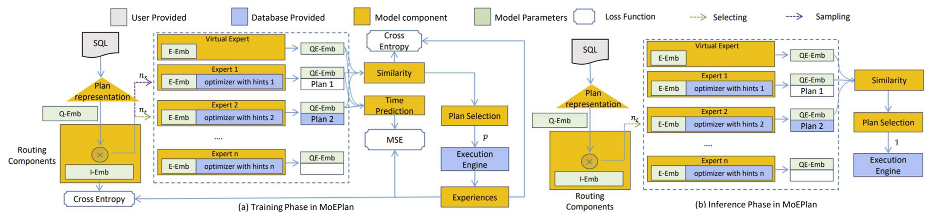


Fig. 2: Framework of MoEPlan

3.2 Query Representation

MoEPlan uses a physical approach for query representation, facilitated by QueryFormer, which employs a transformer to process execution plans. A super node aggregates information from all operators, enabling MoEPlan to estimate execution times for different plans, even if the query embedding is derived from a single plan.

3.3 Routing Component

The routing component selects candidate experts using learned embeddings. Given a query embedding emb_q, expert embeddings emb_e are used to compute an importance vector I_q via a Softmax function as Eq. 1.

$$I_q = \text{Softmax}(emb_e \cdot emb_q) \tag{1}$$

The routing network, which can be pre-trained, is trained on a subset of training queries $Q_r \subset Q$ using a cross-entropy loss as Eq. 2, where $Y_q[i]$ indicates whether the i-th expert produces the most efficient plan.

$$l_{route} = \frac{1}{|Q_r| \cdot |E|} \sum_{q \in Q_r} \sum_{i: e_i \in E} -Y_q[i] \cdot \log(I_q[i]) \tag{2}$$

Top-k experts E_t are selected by the routing network, and additional n_s experts E_s are sampled probabilistically by Eq. 3. The pending expert set $E_p = E_s \cup E_t$ will be considered in the following steps.

$$prob(e_i, q) = \frac{I_q[i]}{\sum_{j: e_j \in E \wedge e_j \notin E_t} I_q[j]}, \text{ if } e_i \notin E_t \tag{3}$$

3.4 Expert and Virtual Ideal Expert Component

Each expert, except the virtual one, includes a prediction network and a built-in optimizer. A feature transformation layer adapts the embedding for prediction. The prediction network estimates execution time t_i from the query embedding emb_q. The MSE loss function Eq. 4 aligns predicted and actual execution times.

$$l_{mse} = \frac{1}{|Q| \cdot |E_p|} \sum_{q \in Q} \sum_{i: e_i \in E_p} (t_q^i - f_{EQ}^i(emb_q))^2 \quad (4)$$

A virtual ideal expert is introduced to provide explicit signals for plan selection. The virtual ideal expert estimates the minimal execution time for each query and measures similarity with other experts using cosine similarity. A softmax function is applied to similarities, and a cross-entropy loss (Eq. 5) is combined with the MSE loss (Eq. 4) using a hyperparameter γ (Eq. 6).

$$l_{virtual} = \frac{1}{|Q| \cdot |E_p|} \sum_{q \in Q} \sum_{i: e_i \in E_p} -Y_q[i] * \log(S_q[i]) \quad (5)$$

$$l_{all} = l_{mse} + \gamma \cdot l_{virtual} \quad (6)$$

3.5 Efficiency Analysis

We compare MoEPlan with other optimizers in terms of *Explain* statements, plan executions, and complexity. During training, Bao requires n *Explain* statements and executes p plans, while Lero explores new plan spaces and also executes p plans. MoEPlan needs only one *Explain* statement and invokes p experts. Bao and MoEPlan use pointwise loss ($O(p)$ complexity), whereas Lero's pairwise loss results in $O(p^2)$ complexity. During inference, Bao and Lero need n *Explain* statements, while MoEPlan needs only one. All methods have $O(p)$ inference time, but Bao and Lero involve n sequential interactions with native optimizers, and Lero's $O(p)$ plan comparisons must be done sequentially.

Table 1: DB Interactions and Time Cost Complexity.

Metrics	Training			Inference		
	Bao	Lero	MoEPlan	Bao	Lero	MoEPlan
The number of Explain Statements	n	n	1	n	n	1
The number of Plan Execution	p	p	p	1	1	1
Time Complexity in Plan Selection Model	$O(p)$	$O(p^2)$	$O(p)$	$O(p)$	$O(p)$	$O(p)$

4 Experiment

In this section, we first outline the experimental setup. Next, we conduct a comparative analysis between MoEPlan and its competitors. Finally, we illustrate the effectiveness of components in MoEPlan through an ablation study.

4.1 Experimental Setting

Dataset. We use two widely used datasets, Join Order Benchmark [6] and TPC-DS [11], in the evaluation of learned plan-selection optimizer.

- **Join Order Benchmark (JOB):** JOB includes 33 templates with 113 queries over the IMDb dataset. Following prior works [1], we use one query per template (1a$\sim$33a) as the test set and the rest for training.
- **TPC-DS:** TPC-DS is a standard benchmark with data and query generators. We generate data at a scale factor of 4. Due to compatibility issues, we use a subset, **TPC-DS-P**, which includes 20 templates with three queries each. We randomly select 40 queries for training and use the rest for testing. For ablation studies, we use **TPC-DS-F**, which includes all queries from TPC-DS, with the first 80 queries for training and the remaining 19 for testing. Some queries in TPC-DS-F are unusually slow due to suboptimal physical operator choices.

Competitors We compare MoEPlan with the following methods.

- **Native Optimizer (NO)**: The default database optimizer.
- **Variant Optimizers (VOs)**: 15 optimizers with different hint combinations based on Bao [7], included in MoEPlan as experts.
- **Optimal Optimizer (OO)**: OO represents the best expert chosen from NO or VOs for each query, defining the upper limit of MoEPlan's capabilities.
- **Bao** [7]: A learned plan-selection method using hints to generate diverse plans and a value model to estimate costs, modeled as a multi-armed bandit problem, with the Thompson sampling method employed to balance exploration and exploitation.
- **Lero** [19]: Lero extends Bao by focusing on relative plan performance and using row number corrections for candidate plan generation.
- **FASTgres** [14]: FASTgres is a learning-based classification strategy that partitions queries into contexts and trains models to map predicates to optimal hints.

Evaluation Metrics. We use two metrics to evaluate model performance:

- **Speed Up (SU):** Measures the overall acceleration in execution time relative to NO.
- **Geometric Mean Relative Latency(GMRL):** Focuses on the acceleration ratio of each query.

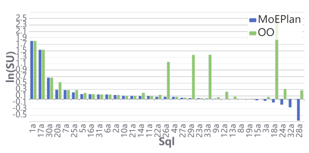
4.2 Overall Performance of MoEPlan

We first study the overall performance comparison of varying datasets. Then, we investigate the performance improvement on JOB, TPC-DS-F and TPC-DS-P. Finally, we compare the inference time and execution time of MoEPlan with those of competitors.

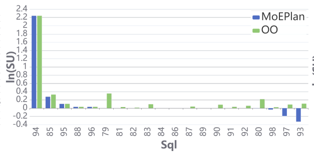
Overall Comparison of Varying Datasets. Table 2 compares competitors across different datasets. NO is the baseline optimizer. To ensure fairness, both FASTgres and Bao can choose from 16 hint sets. FASTgres is excluded from TPC-DS-F and TPC-DS-P due to lack of support [14], and Bao and Lero are similarly excluded from TPC-DS-F.

Table 2: Performance Comparison of Varying Datasets.

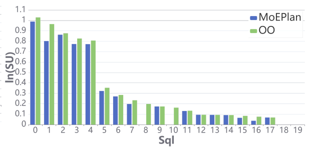
Model	JOB		TPC-DS-F		TPC-DS-P	
	SU	GMRL	SU	GMRL	SU	GMRL
NO	1.00	1.00	1.00	1.00	1.00	1.00
Best of VOs	1.20	0.87	2.45	1.00	1.60	0.79
Bao	1.32	0.88	-	-	1.75	0.81
Lero	1.21	0.87	-	-	1.26	0.85
FASTgres	1.39	0.86	-	-	-	-
MoEPlan	**1.50**	**0.86**	**2.71**	**0.91**	**1.82**	**0.75**



(a) JOB Test Sets



(b) TPC-DS-F Test Sets



(c) TPC-DS-P Test Sets

Fig. 3: Comparison of MoEPlan and OO in Terms of SU on JOB, TPC-DS-F, and TPC-DS-P Test Sets

As shown in Table 2, MoEPlan integrates the strengths of different optimizers and achieves the best performance. This demonstrates MoEPlan's ability to optimize performance and select efficient experts. A more detailed analysis of MoEPlan's expert selection will be provided in the performance breakdown for each query.

Performance Study on JOB, TPC-DS-F, and TPC-DS-P Test Sets. We calculate the natural logarithm of SU for MoEPlan and the OO for each query across test sets.

For the JOB test sets (Figure 3a), MoEPlan's SU logarithm often exceeds 0, showing better performance than NO. In the TPC-DS-F test set (Figure 3b), MoEPlan matches the optimal choice in most cases. For TPC-DS-P (Figure 3c), MoEPlan outperforms NO and frequently achieves optimal expert selection.

Inference Time and Execution Time. This study focuses on the inference time of query optimizers, which is the time taken to generate an execution plan and is crucial

Table 3: Total Inference and Execution Time on JOB and TPC-DS-P Test Sets

Model		NO	MoEPlan	Bao	Lero	FASTgres
JOB	$t_i\ (ms)$	**943**	3108	16901	204019	4211
	$t_e\ (ms)$	44573	**29709**	34991	36719	32170
TPC-DS-P	$t_i\ (ms)$	**161**	595	1589	3551	-
	$t_e\ (ms)$	22617	**12405**	12895	18026	-

Table 4: Ablation Study of MoEPlan

Model	Model Components			JOB		TPC-DS-F	
	Routing Network	Pre-training	Virtual Ideal Expert	SU	GMRL	SU	GMRL
MoEPlan		✓	✓	1.036	0.965	2.677	0.979
	✓		✓	1.114	1.063	**2.722**	0.926
	✓	✓		1.369	0.878	2.697	0.969
	✓	✓	✓	**1.500**	**0.862**	2.714	**0.908**

for our design, excluding the execution time. Table 3 compares the inference (t_i) and execution (t_e) times for all competitors on JOB and TPC-DS-P datasets.

MoEPlan demonstrates lower inference and execution times compared to Bao, Lero, and FASTgres. While learned optimizers typically have higher inference times, MoE-Plan's is higher than NO's due to its two-stage plan generation process. Bao and Lero's high inference times result from multiple optimizer invocations, and FASTgres's is due to the need to extract query context.

4.3 Effectiveness of Components

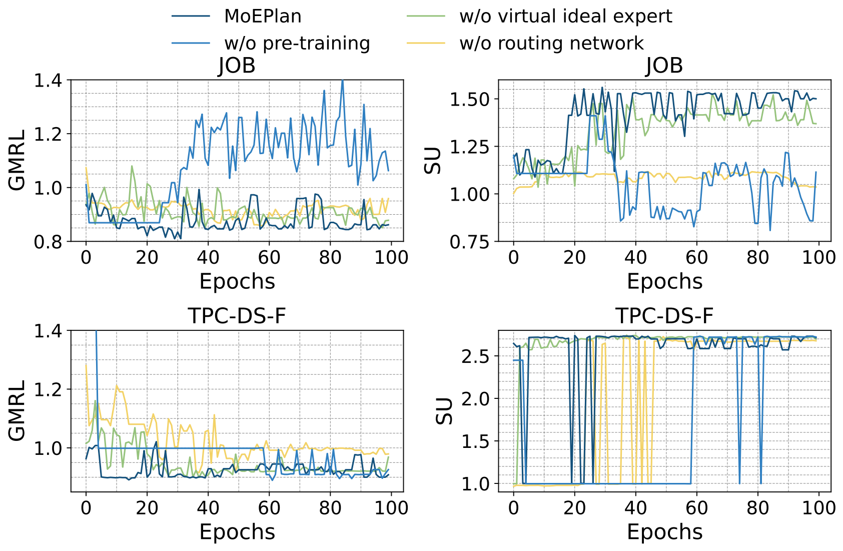


Fig. 4: GMRL and SU Metrics Comparison on JOB and TPC-DS-F

To thoroughly assess the factors affecting MoEPlan's performance, we perform a series of ablation studies on MoEPlan's components.

Role of Routing Network. Removing the routing network severely degrades MoE-Plan's performance (Figure 4, Table 4). The routing network is necessary for identifying efficient experts and simplifying the task of the expert network. Without routing network, MoEPlan focuses on slow experts, highlighting its necessity.

Role of Pre-training. Without pre-training, performance drops significantly (Figure 4). The routing network may select incorrect experts, destabilizing the model. For example, MoEPlan fails to achieve GMRL below 1 on JOB, showing ineffective routing.

Role of Virtual Ideal Expert. The virtual ideal expert ensures a stable plan selection (Figure 4). The absence of the virtual ideal expert causes performance fluctuations and biases toward long queries, especially on TPC-DS-F. The virtual ideal expert is essential for consistent results across datasets.

5 Related Works

Besides the database hints and gating network mentioned in Section 3, related works include query representation and learned database optimizer.

Representation of Execution Plan. Execution plan representation is crucial for tasks like cardinality/cost estimation. The early methods used operator-level neural units for different operators, assembled according to the plan structure [9]. Tree-structured models like Tree-LSTM [12] and Tree-CNN [10] aggregate information from leaf nodes along tree paths. QueryFormer [18] extends transformers for query plan representation, enabling handling of diverse queries.

Learned Database Optimizer. Plan-selection optimizers, including Bao and Lero, have been extensively discussed. FASTgres [14] partitions query workloads and predicts hints using Gradient Boosting algorithm. However, FASTgres may struggle with changing workloads. Plan-generation optimizers include RTOS [17], which uses reinforcement learning with Tree-LSTM for subtree evaluation, and LOGER [1], which extends RTOS with beam search and combined loss functions. Balsa [16] employs policy-based reinforcement learning, trained without expert demonstrations.

6 Conclusion

In this paper, we propose MoEPlan, a learned plan-selection method, to lower the time consumed in the optimized plan generations. MoEPlan incorporates different optimizers as experts and designs a routing network to match the incoming query with candidate experts. MoEPlan then learns an expert network to estimate execution time and finally uses the virtual ideal expert to explicitly guide the ranking of the experts, selecting the best expert to evaluate the query. Experimental results demonstrate the effectiveness of the generated plans and the efficiency of MoEPlan.

Acknowledgemengts

This work is supported by NSFC (No. 62272008) and ZTE-PKU joint program.

References

1. Chen, T., Gao, J., Chen, H., Tu, Y.: Loger: A learned optimizer towards generating efficient and robust query execution plans pp. 1777–1789 (2023)
2. Chen, Z., Deng, Y., Wu, Y., Gu, Q., Li, Y.: Towards understanding the mixture-of-experts layer in deep learning. In: NeurIPS 2022 (2022)
3. Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: scaling to trillion parameter models with simple and efficient sparsity pp. 5232–5270 (1 2022)
4. Ibaraki, T., Kameda, T.: On the optimal nesting order for computing n-relational joins pp. 482–502 (9 1984)
5. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. Neural Comput. (1), 79–87 (1991)
6. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T.: How good are query optimizers, really? pp. 204–215 (11 2015)
7. Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., Kraska, T.: Bao: Making learned query optimization practical. In: SIGMOD 2021. pp. 1275–1288
8. Marcus, R., Negi, P., Mao, H., Zhang, C., Alizadeh, M., Kraska, T., Papaemmanouil, O., Tatbul, N.: Neo: A learned query optimizer (2019)
9. Marcus, R., Papaemmanouil, O.: Plan-structured deep neural network models for query performance prediction. Proc. VLDB Endow. **12**(11), 1733–1746 (2019)
10. Mou, L., Li, G., Zhang, L., Wang, T., Jin, Z.: Convolutional neural networks over tree structures for programming language processing. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 1287–1293 (2016)
11. Poess, M., Smith, B., Kollar, L., Larson, P.: Tpc-ds, taking decision support benchmarking to the next level. In: SIGMOD. pp. 582–587 (2002)
12. Tai, K.S., Socher, R., Manning, C.D.: Improved semantic representations from tree-structured long short-term memory networks. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics. pp. 1556–1566 (2015)
13. postgresql team: pg-hint-plan 1.6, `https://pg-hint-plan.readthedocs.io/en/latest/`
14. Woltmann, L., Thiessat, J., Hartmann, C., Habich, D., Lehner, W.: Fastgres: Making learned query optimizer hinting effective. Proceedings of the VLDB Endowment **16**(11), 3310–3322 (2023)
15. Yan, Z., Uotila, V., Lu, J.: Join order selection with deep reinforcement learning: Fundamentals, techniques, and challenges pp. 3882–3885 (8 2023)
16. Yang, Z., Chiang, W.L., Luan, S., Mittal, G., Luo, M., Stoica, I.: Balsa: Learning a query optimizer without expert demonstrations. In: Proceedings of the 2022 International Conference on Management of Data. pp. 931–944 (6 2022)
17. Yu, X., Li, G., Chai, C., Tang, N.: Reinforcement learning with tree-lstm for join order selection. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE). pp. 1297–1308. IEEE (2020)
18. Zhao, Y., Cong, G., Shi, J., Miao, C.: Queryformer: A tree transformer model for query plan representation. Proc. VLDB Endow. **15**(8), 1658–1670 (2022)
19. Zhu, R., Chen, W., Ding, B., Chen, X., Pfadler, A., Wu, Z., Zhou, J.: Lero: A learning-to-rank query optimizer pp. 1466–1479 (2023)