Summary of Changes — LOGER: A Learned Optimizer towards Generating Efficient and Robust Query Execution Plan

Tianyi Chen
Key Laboratory of High Confidence Software Technologies, CS, Peking University, China
tianyichen@stu.pku.edu.cn

Jun Gao
Key Laboratory of High Confidence Software Technologies, CS, Peking University, China
gaojun@pku.edu.cn

Hedui Chen
ZTE Corporation, China
chen.hedui@zte.com.cn

Yaofeng Tu
ZTE Corporation, China
tu.yaofeng@zte.com.cn

Thanks for providing us with the opportunity to submit a revised version of our manuscript. We deeply appreciate the detailed and constructive comments. According to these comments, we work hard and make a major revision to extend the techniques, improve the presentation, and perform extensive experiments to verify our claims in the last three months. Specifically, we conduct new experiments from different viewpoints, including large underlying datasets, more competitors, more ablation study experiments, and different training/testing workload splitting strategies. We believe that we have addressed all concerns from reviewers, and expect that the findings of our research starting from August 2021 with heavy development work can finally contribute to the database communities.

In the following contents, we first summarize the major changes of the revised version and then answer issues raised by reviewers one by one.

Technical Extension. In the revised version, we mainly extend LOGER to support schema changes. The previous version of LOGER only supports a fixed number of tables/columns, and the schema changes require the adjustment of source codes and model retraining. In contrast, the revised LOGER can handle updated schema by incremental model training without a change of source codes, like RTOS.

- 1. We have redesigned LOGER to avoid the use of one-hot encodings in the previous version. For the one-hot encodings in query representation, we replace them with learned embeddings of each table. We also avoid column-level learned matrices in the previous version by table-level matrices and statistics-based encoding vectors, as the previous version causes generalization problems of unseen columns in testing workload. In addition, we have modified the operator representation of the value model to avoid fixed-size vectors related to the number of tables.
  Since one-hot encodings no longer exist in our method, LOGER now supports incremental training to fine-tune the trained parameters for updated schema rather than adjusting the source codes and retraining the model from scratch. We have conducted experiments on LOGER over updated schema following RTOS. The results show that LOGER successfully fine-tunes on the previous checkpoint to support plan generation of the additional table's queries.
- 2. We have simplified the design of value model by removing a number of redundant inputs. The extra representation vector e in the previous version is now replaced by aggregated learned representations for join operators on each table. The adjacency matrix and selectivity vector inputs are removed as the query representation module already captures the information.

Experiment. We have made extensive experimental studies and reported results in section *Experiments* to demonstrate the effectiveness of LOGER.

- 1. We have collected and reported training and inference time cost for LOGER and competitors, and found that LOGER has a relatively low inference time compared with Bao, RTOS and Balsa. Bao runs fastest in the training phase.
- 2. We have added Bao as a competitor in our experiments. The reason that we did not compare LOGER with Bao in the previous version is that we failed to reproduce its results. After careful checking, we finally corrected our mistakes in running Bao and now the results are in the section *Experiments*. Through comparison, we can see that LOGER can generate more efficient execution plans with less inference time, at the expense of a higher training time compared with Bao.
- 3. We have conducted experiments on a large dataset, Stack Overflow (also used in Bao's paper), following the suggestion of Reviewer 2. The results shows that LOGER has a high performance on Stack Overflow workload.
- 4. We have added a simple baseline for comparison, in which the merge join operator is not allowed in PostgreSQL, denoted as PG-NoMerge. We can see that PostgreSQL can yield high performance on Stack Overflow workload when merge join is disabled, while such a restriction does not work on other workloads. We have also tested disabling nested loop join operator or hash join operator, and received worse performance. Globally disabling nested loop join is especially disastrous on JOB workloads. Due to the limited space, we do not report the results in the paper.
- 5. We have conducted experiments using another training/testing workload splitting strategy on JOB used in Balsa, on which the plans generated by LOGER still perform best.

In addition, the training and testing workloads are divided by templates on both TPC-DS and Stack Overflow in the revised version.

- 6. We have made a study to show which strategies alleviate the cold start problem. There are multiple strategies that may help LOGER to quickly reach higher performance and become stable, including 1) ϵ-beam search, 2) the reward weighting strategy (we called it *weight transformation* in the previous version), 3) the restriction of producing left-deep plans, 4) using the plans generated by DBMS built-in optimizer as expert knowledge at the beginning of training.
- 7. We have performed studies on different pooling strategies in the query representation module of LOGER, and the results are reported in the subsection of the ablation study. We found that max-pooling produces the best WRL and training stability but the GMRL with sum-pooling and mean-pooling is slightly better. We guess that through max pooling, the query representation module pays more attention to columns with the largest selectivities but tends to ignore information from other columns.

Presentation Improvement. We have reorganized the paper to improve the presentation and clarify our contributions.

- 1. We have added a symbol table in the paper as suggested by Reviewer 1.
- 2. We have rewritten most sections of the paper. The section *Method Framework* is reorganized to explain LOGER in a self-contained way. Specifically, we redraw Figure 1 and highlight the data provenance of value-based reinforcement learning in LOGER. We explain the entire data processing in LOGER using the example in Figure 1, and discuss the relationships of the following three sections. We largely modified the section *Query Representation* because of the technical extension. The weight transformation method in the previous version is renamed as *reward weighting*, and the misleading terms, operator-aware and order-aware feedback, are renamed as operator-relevant and operator-irrelevant latency. The example of ϵ-beam search in the section *Plan Search* is modified to clarify its procedure, along with its novelty and advantage.
- 3. We have added a table in *Ablation Study* for the comparisons of LOGER with other competitors to give a comprehensive view from different viewpoints. In addition, we have redrawn and discarded some figures of experimental results as suggested by reviewer 3.
- 4. We have revised the paper carefully, including i) correcting grammatical errors and typos in the statement, and ii) making more explanations for the figures. We also asked other English speakers to proofread the paper. We believe that the revised version is clearer than the previous one.

Code Availability. After discussing with the funding providers, we are allowed to upload the codes of LOGER as complementary material to fulfill our promise of open-sourcing in the previous submission. The codes can now be used to verify the performance.

In the following content, we address issues raised by each reviewer one by one.

RESPONSE TO META-REVIEWER

(1) **Comments**: *The paper proposes LOGER, a combination of Graph Transformer with deep reinforcement learning (DRL) method for efficient and robust plan generation. The reviewers agree that the contributions of this work are relevant, yet they would like to see a revised version of this work that will address several important questions, as detailed under revision items..*

(2) **Revision Items**: *1) Add an analysis of the training cost, inference time, and query performance across different workloads and dataset sizes for all competitors listed in Table 1.*
Response: We have reported training time cost and inference time cost along with different measurements for query execution performance (WRL and GMRL) across different workloads for all competitors, listed in Table 2. We are delighted to see that LOGER requires a short time in plan inference and query execution, while Bao takes the shortest time in the training phase.

(3) **Revision Items:** *2) Explain how the model generalizes to updated schemas or different databases.*
Response: Thanks! Please check item 1 in *Technical Extension* in the summary part.

(4) **Revision Items** *3) Explain how the cold start problem is avoided.*
Response: There are different strategies in LOGER that may alleviate the cold start problem. We have conducted experiments in ablation study to see the effects of these strategies in section *Experiments*, and found that the following strategies majorly contribute to the problem.

1. ϵ-beam search. ϵ-beam search keeps multiple promising sub-plans during plan search and performs exploration guided by the value model's prediction. Hence, the plan samples generated by ϵ-beam search have more chances to be efficient. These high-quality samples enable LOGER to learn quickly and stably. In contrast, LOGER is unable to reach convergence if ϵ-greedy is used as an alternative to ϵ-beam search.

2. The reward weighting strategy. The plans generated by LOGER include both join order and (restricted) operators. As two factors are interacted, plans with identical join orders but different operators may have completely different performances, leading to severe fluctuations in reward values. LOGER uses reward weighting to reduce the fluctuation, enhancing stability and also encouraging the model to learn join order selection quickly.

3. The restriction of producing left-deep plans. We use left-deep restriction in experiments of JOB and TPC-DS. With left-deep restriction, the search space is significantly reduced and thus it is relatively easy to learn how to produce efficient plans. It can be seen from ablation study that allowing bushy plans on JOB causes difficulty in cold-starting and leads to unstable performance.

4. Introducing expert knowledge at the beginning of training. The query execution plans from DBMS built-in optimizer are added into experience dataset initially, which can provide the learned optimizer with high-quality training samples. Such a strategy forces LOGER towards the plans generated by DBMS, and lowers the chances for LOGER to generate poor plans when training from scratch.

(5) **Revision Items**: *4) Improve the presentation: the explanation of the loss function, the provenance of data points, the division between training and test set, the novelty over prior work.*
Response: We have revised the paper to address these concerns. The revisions are explained as follows.

Explanation of the loss function. First, we simplify the loss function in the revised LOGER. The previous loss function includes a multi-task strategy that uses both state value function $V(S;\theta)$ and state-action value function $Q(S,a;\theta)$. After ablation study, we found that keeping only the state-action value function can achieve similar results. Hence, we remove the multi-task mechanism in the revised version.
We use reward weighting (named as weight transformation previously) to obtain ground truth values for $Q(S,a;\theta)$, followed by log transformation. Reward weighting combines operator-relevant latency and operator-irrelevant latency together to reduce value fluctuation. Denote $S=(J,O)$ where J represents the join order and O represents the operators. The operator-relevant latency $T((J,O),a)$ is the reached lowest latency of (S,a), which is widely used as ground truth value in other DRL optimizers. The operator-irrelevant latency $T((J,/),a)$ is the minimum value of all $T((J,O'),a)$ that share the same join order J and action a with (S,a). Reward weighting combines the two values by weighting to produce ground truth values, in order to stabilize value fluctuation caused by operator selections. It also leads LOGER to mainly focus on join order selection. In addition, the log transform function $\mathrm{LT}(x)$ compresses the wide range of disastrous rewards to reduce model sensibility to these values, and let LOGER pay more attention to efficient plans.

The provenance of data points. It is related to the settings of value-based reinforcement learning. We redraw Figure 1, the framework of LOGER, and add related descriptions to highlight the provenance of data points. As sub-plans (① - ③ in the figure) cannot be evaluated alone, their values depend on the final plan (④). When the final plan is produced, for each of those subplans $S=(J,O)$ and corresponding action a, LOGER updates the table value $T((J,O),a)$ in the experience dataset, which represents the minimum latency of all final plans that share the same intermediate (S,a). To perform reward weighting, the values of $T((J,/),a)$ are also updated through this process.

The division between training and test set. Apart from our previous division strategy, we conduct experiments on JOB according to the division strategy used by Balsa, which is a random splitting strategy by instance. In both cases, LOGER generates plans with higher performance than the competitors. In addition, the training and test set are divided by templates on both TPC-DS and Stack Overflow in the revised version.

The novelty over prior work. We summarize and focus on three major contributions in LOGER: i) a novel search space, ROSS, which can leverage the existing DBMS built-in optimizer to select physical operators while preserving exploration capability; ii) a new search method, ϵ-beam search, which introduces exploration into beam search with the idea of ϵ-greedy, aiming at finding efficient plans to obtain high-quality training samples; iii) reward weighting, which combines operator-relevant and operator-irrelevant latency to further improve stability and encourage the model to learn efficient join order.

(6) **Revision Items**: *5) Add experiments with larger and more diverse datasets such as the Stack Overflow workload.*
Response: We have added Stack Overflow dataset (used by Bao) in experiments. It can be seen that the plans generated by LOGER also have performance improvement on Stack Overflow compared with the plans generated by other competitors.

(7) **Revision Items:** *6) Add experiments with simple baselines, like disabling problematic operators or using a robust left-deep ordering.*
Response: We disable the merge join operator and then use the generated execution plans of PostgreSQL as a baseline, named as PG-NoMerge. We have also tested disabling the nest loop join operator and disabling the hash join operator, and found that PG-NoMerge produces query plans with the stablest performance, while disabling nested loop join or hash join operator leads to a performance decrement on JOB, with no advantage on other workloads. Limited by space, we only demonstrate the performance of PG-NoMerge in the paper. The results illustrate the importance of a large search space to find efficient plans.
Honestly, for the term *robust left-deep ordering*, we do not understand its meaning, even if we search the term via Bing and Google. We guess it is a kind of heuristic method that specifies left-deep join orders. If so, we will take it as a baseline in future work.

(8) **Revision Items** *7) A comparison against Bao would be appropriate, as the two approaches are similar in spirit.*
Response: We have added Bao to the experiment part. The reason that Bao is not included in the previous version is explained above. With the experiment results from Bao, we can see that LOGER produces plans with better performance and generally lower inference time, although LOGER requires a longer training time.
We try to state the similarity and differences between Bao and LOGER as follows. Bao generates execution plans using the DBMS built-in optimizer with different settings of global parameters. Inspired by Bao, LOGER tries to utilize the expert knowledge of DBMS. Specifically, LOGER searches candidate plans in Restricted Operator Searching Space (ROSS), in which the DBMS optimizer selects proper join operators under specified restrictions produced by LOGER.

Meanwhile, Bao does not generate plans itself. Bao first invokes the built-in optimizer to generate a plan for each arm, i.e. group of global parameter settings, and then selects among generated plans. In contrast, LOGER first generates the plan by itself, which includes both join order and restricted operators, and then invokes the built-in optimizer to further decide physical operators. The key difference is that LOGER's search space is much larger, with more chances to generate better plans, while Bao can only selects from a fixed number of plans.

(9) **Revision Items**: *8) DRL is notoriously sample inefficient. How bad is performance at iteration zero, and what new techniques do you offer to improve/alleviate this?*
Response: DRL-based learned optimizer usually requires a large number of samples before producing an efficient evaluation plan. LOGER attempts to alleviate the issue by finding high-quality samples, so as to allow the model to achieve higher performance with fewer samples. The following strategies are used in LOGER.
First, we leverage the existing DBMS built-in optimizer to generate initial data into experience dataset. In the initial stage of plan training, the plans generated from the DBMS optimizer are used to train the value model of LOGER by sampled mini-batches. Meanwhile, LOGER introduces ROSS, the restricted operator search space, in which DBMS optimizer supports LOGER to select proper physical operators in the plan, making it much easier to generate efficient plans.
Second, we design an efficient plan search strategy for finding high-quality samples of subplans. ϵ-beam search can perform a wider search than ϵ-greedy and keep multiple promising subplans. In addition, the exploration paths are randomly selected but eventually controlled by the value model to perform investigation along promising directions, which increases the effectiveness of samples.
Third, LOGER restricts the form of samples as left-deep trees in the first 4 epochs (even if bushy plan generation is finally allowed) in all experiments. The restriction significantly reduces the search space but can result in efficient plan generation.

(10) **Revision Items:** *9) It is critical to customize a neural architecture to the underlying problem (i.e., create a good inductive bias). How do you do this? Can you intuitively explain the bias created?*
Response: Most existing learned optimizers follow the reinforcement learning framework. After converting the query into vectorized representation, some of them like DQ, RTOS and Balsa build a value model with MLP, Tree-CNN, or Tree-LSTM, etc. to evaluate different states (subplans) in the search procedure, while some others like ReJoin directly learn the policy of action (join) selection. In our opinion, the existing learned optimizers share similar neural architectures of plan representation, but the methods to extract information from queries are different, leading to different expressiveness. For the network, we introduce Graph Transformer into LOGER to further utilize the structure of join graph to extract relationships of tables and predicates, and show the effectiveness of GT by ablation study.
Meanwhile, some characteristics of query optimization distinguish the DRL optimizers from reinforcement learning methods of general purposes, which have shown an impact on sample generation, exploration and obtaining reward values. It is of great importance to solve these problems as well as use a proper network. LOGER deals with these problems through other components including ROSS, ϵ-beam search and reward weighting, etc.

(11) **Revision Items** *10) What is the "regression" story? What is the intuition behind the new techniques proposed that prevent the optimizer from going off the rails? How does the optimizer respond to change?.*
Response: The following components or strategies help LOGER to avoid generating poor plans for unseen queries.
First, the ϵ-beam search in LOGER helps find subplans with high potential. The subplans that are likely to have poor performance are discarded in the search process. During inference, beam search is also utilized to generate promising plans, with the exploration disabled.
Second, ROSS decreases the difficulty of finding efficient plans with ROSS. By leveraging the knowledge of built-in optimizer, LOGER learns to select proper operators easier in ROSS, which further prevents the generation of poor plans.
Third, the revised LOGER supports incremental training. When the schema is updated or the underlying data distribution is changed, incremental training can train the parameters of new tables while fine-tuning that of other tables and learned parameters of the network, enabling LOGER to fit into the new context.
In the future, we will study on utilizing the advantages of data-driven methods, so as to further improve the generalization ability of LOGER.

(12) **Revision Items**: *11) What baselines do you choose and why are your selected baselines enough to demonstrate that your technique is effective?*
Response: We compare our method with RTOS, Bao and Balsa, which are representative works of learned query optimization, published by ICDE 2021, SIGMOD 2021 (Best paper), and SIGMOD 2022, respectively. In addition, we compare LOGER with a simple baseline PG-NoMerge (PostgreSQL with merge join operator disabled) and a Commercial DBMS.

RESPONSE TO REVIEWER-1

(1) **Summary**: *The manuscript proposes LOGER which combines Graph Transformer with deep reinforcement learning (DRL) method for efficient and robust plan generation. They define for types of restricted operator in Restricted Operator Search Space (ROSS) to reduce the risk of poor plans caused by direct selection. After that, the authors combine beam search and ϵ-greedy to handle the exploration and exploitation trade-off. An experimental study is conducted to demonstrate the effectiveness of LOGER.*

(2) **Comments**: *W1: Most of the figures in the manuscript need more specific descriptions.*
Response: Thanks for your suggestion. We have redrawn most figures in the manuscript, and added more specific descriptions about the figures.

(3) **Comments**: *W2: The overall framework is not clearly described in the manuscript. There is lack of coherence between sections.*
Response: Thanks for your suggestion. We have rewritten *Method Framework* to clarify our design. Specifically, we redraw the framework in Figure 1 and highlight the data provenance of value-based reinforcement learning in LOGER. We explain the entire data processing procedure in LOGER using the example in Figure 1, and discuss the sequential relationship of the following three sections.

(4) **Comments**: *W3: It would be better to provide a notation table in the manuscript to specify the variables.*
Response: Thanks for your suggestion. We have added a notation table in the manuscripts.

(5) **Comments**: *D1. The paper has some grammatical errors and typos, such as "A number of existing learned query optimizers..., which provides less chance to achieve better performance.", formula (5.2) in section 5.2, titles of section 4.3 and 5.3, IMDb in section 6.1.*
Response: Thanks. We have revised the paper carefully and corrected these and other errors in the manuscripts.

(6) **Comments**: *D2. The paper did not clearly articulate the novelty of its approach and how it differs from existing works. The example of ϵ-beam search did not present the advantage with ϵ-greedy*
Response: We have rewritten the section *Introduction* and *Method Framework* to clarify our novelty over previous methods. Please refer to feedback to Item 4 of the meta-reviewer for the novelty of the paper. We redraw the figure about ϵ-beam search. We stress the difference between ϵ-greedy and ϵ-beam search in two aspects in the revised version.
First, there is only one search path in linear search methods including ϵ-greedy. When the value model selects the next action by comparing different state-action pair candidates, these pairs are enumerated from the same state. This comparison is then highly localized and greedy, incurring inefficiency in finding globally good plans in the search space. In contrast, ϵ-beam search keeps K search paths, and thus state-action pairs from multiple regions of the space are compared, providing wide visions to LOGER.
Second, unlike the fully random selection of exploration in ϵ-greedy, the exploration in ϵ-beam search selects these paths through value prediction. Specifically, LOGER randomly selects a number of paths from state-action pair candidates of exploitation paths (except the candidates already selected by exploitation), and then determines exploration paths by top-k selection using the value model. In this way, LOGER restricts the exploration towards a potentially better direction, as the search space is too large and fully random selection usually obtains poor results.

(7) **Comments**: *D3. The symbol of some variables is not clearly defined. "Excluding T-candidates, we randomly choose$\lambda \times k_r$ state action pairs from current T-paths...", what does λ mean. The symbol of execution latency is Latency($\cdot$) or T(Si).*
Response: Thanks. We have removed some misleading terms and improved the presentation of the paper.

(8) **Comments**: *D4. The experiments do not include the time consuming of model training*
Response: Thanks! Please refer to feedback to Item 1 of the meta-reviewer.

RESPONSE TO REVIEWER-2

(1) **Summary**: *The authors propose LOGER, a new learned query optimizer. Unlike prior work, LOGER uses a novel representation of the data schema (including predicates, columns, etc.) and a representation learned via GA. Additionally, LOGER works side by side with a traditional query optimizer by restricting the search space LOGER considers based on the plan chosen by the optimizer. Experimentally, the authors show LOGER can outperform PostgreSQL by 2x.*

(2) **Comments**: *W1. Several parts of the design of LOGER seem under-explained.*
Response: Thanks! Please check the feedback to W2 of Reviewer 1.

(3) **Comments**: *W2. The experimental results lack suitable baselines.*
Response: We add two baselines, including a simple method that disables the merge join operator of PostgreSQL (named as PG-NoMerge), and Bao, as is explained in item 2 and item 4 in the experimental part of the summary. The simple baseline achieves good performance on the newly added Stack Overflow workload, but has a low effect on other workloads.

(4) **Comments**: *W3. The experiment analysis only looks at one real-world(ish) dataset.*
Response: Thanks! Now we have three datasets, including IMDB (JOB), TPC-DS, and Stack Overflow (newly added in the revised version), as is explained in item 3 of the experimental part of the summary.

(5) **Comments**: *D1) Sec 1. Considering N operator types at each node of the plan tree increases the space multiplicatively, not exponentially.*
Response: Thanks! We avoid the term *exponentially* in the revised version. Originally, we assume that for k logical join nodes that each has choices of N operators, the total number of valid operator combinations is N^k.

(6) **Comments**: *D2) The intro should make it clear what components of the query plan LOGER generates – is it only operator selection (ROSS)? or does it include access operators and join order?*
Response: In the revised version, we clearly state that LOGER focuses on join plan generation including both join order and (restricted) operator section (ROSS). LOGER does not take the access operators (scan methods) into consideration, which is one of our future works.

(7) **Comments**: *D3) Using a GT on the table/key/column graph is really smart! Previous approaches using TCNN for query plan trees seem to use pretty arbitrary representations of the predicates and schema. I also like the idea of a "learned matrix" to represent various data items, kind of like an embedding that can co-learned with the query optimizer.*
Response: Thanks for your kind comments!

(8) **Comments**: *D4) 3.2 typo, capitalize the L in Laplacian*
Response: Thanks! We have corrected the error.

(9) **Comments**: *D5) The discussion of beam search with ROSS and DOSS seem to suggest that the primary area in which QO "goes wrong" is in operator selection. I think past research (i.e., "How Good are QUery Optimizers Really") indicates that join ordering is normally a dominate factor. Why focus on operators instead of ordering?*
Response: Thanks for your comment. LOGER generates plans with both join order and (restricted) operators. We think both join order and physical operators are important for performance. We agree that the join order is a dominant factor. However, even if the join order is optimal, improper selection of physical operators can also severely degrade execution performance. For example, sort-merge join is not frequently used in efficient plans of Stack Overflow queries, and such a statement is verified by the experimental results of disabling the merge join operator in PostgreSQL. Besides, we can see that join order and physical operators interact. For example, Bao invokes DBMS optimizer to generate plans with global settings that disable specific physical operators, and we can see that the join orders of generated plans are sometimes different under distinct physical operator settings.
We have to say that the design of LOGER considers join order selection prior to (restricted) operator selection. With the reward weighting mechanism (named as weight transformation in the previous version) of the loss function, the plans with correct join order but improper operator selections are encouraged by combining the operator-relevant and operator-irrelevant latency values (named as operator-aware and order-aware feedback previously) as rewards.

(10) **Comments**: *D6) I appreciate the explicit discussion of balancing exploration and exploitation. Many past papers are especially vague on this point.*
Response: Thanks for your kind comments! The ϵ-beam search in LOGER has its advantages over traditional ϵ-greedy, as is discussed in feedback to D2 from reviewer 1.

(11) **Comments**: *D7) In 5.1, I do not understand how C is computed. How do you know how long the DBMS optimizer query takes?*
Response: We have modified our description on C (renamed as l) in the revised version. We directly execute the query without explicitly obtaining the plan generated by DBMS optimizer to get the execution latency, since the plan is already implicitly used by DBMS during execution.

(12) **Comments**: *D8) I think almost all of prior works – Neo, Balsa, RTOS – actually use the log of query latency in their implementations, even if they don't mention it. Is the learnable parameter really all that relevant?*
Response: We totally agree with your comments. The query latency varies sharply which significantly impacts the model training stability. We make an analysis of the source codes of Bao and RTOS, and find that these methods also use functions similar to the log transformation function $\mathrm{LT}(\cdot)$ in our method, to compress the range of query latency.

(13) **Comments**: *D9) The restriction to left-deep plans seems quite extreme! In such an arrangement, it seems like 1D convolution might be the best possible model, since everything is essentially a line.*
Response: We agree that 1D convolution actually works on left-deep plans. LOGER also supports the generation of bushy plans, since some workloads like the newly added Stack Overflow workload may require bushy plan structures to reach high performance. When the query plan has a more general shape, such as a bushy structure, it is better to use more powerful models like Tree-LSTM, to capture the information of the plan. To support the generation of bushy structured plans, we choose Tree-LSTM for state representation in LOGER. We show the effect of the left-deep restriction on JOB by ablation study in the paper.

(14) **Comments**: *D10) I do not understand the "operator" and "ordering" losses. Are these different Q functions estimating the best possible operator selection and best possible ordering? Or are they ground truth? Where does that data come from?* .
Response: Sorry that we give a misleading term. Originally, these terms are related to different rewards (ground truth values), which are combined together by weighting in the loss function. Your comment inspires us to rename terms into operator-relevant and operator-irrelevant latency.
Please further check the feedback to item 4 of the meta-reviewer.

(15) **Comments**: *D11) Experimental datasets are quite limited. JOB is good, but is a very small database. Why do you only test on 20 TPC-DS queries instead of all 99? Additionally, consider adding a larger more "realistic" workload like the Stack Overflow workload used in Bao.*
Response: Thanks for your suggestion. We now add Stack Overflow workload into the experiments.
LOGER focuses on select-project-join queries. The reason that we use 20 TPC-DS templates instead of 99 is that most templates of TPC-DS do not generate select-project-join queries, mostly containing sub-queries. We also discard templates 9 and 10 of Stack Overflow for the same reason. In some cases, sub-queries can be transformed into equivalent select-project-join queries, but this is out of the scope of this paper.

(16) **Comments**: *D12) Analysis of metrics like WRL and GMRL are great for looking at overall performance, and the tail CDFs are nice but still leave things in relative terms. Most of the queries get better, but do some queries get much much worse? Plotting the 95/99/99.9 percentiles in real terms can be helpful to see this, or you can plot the performance of every JOB query sorted by improvement.*

Response: Thanks! We plot the speedup of each JOB query compared with PostgreSQL sorted by improvement in Figure 9 in the revised version following your valuable and constructive comments.

(17) **Comments**: *D13) If you are going to talk about workload level descriptors, plot the absolute, not the relative, difference – a 10x speedup on a 100ms query does not make up for a 2x slowdown on a 10 minute long query!*

Response: We follow your advice and redrew the figures. Figure 9 now shows the absolute improvements on each JOB query, from which it can be seen that on testing workload, LOGER achieves improvement greater than 1000ms for 12% of queries, with 6% of queries inferior to PostgreSQL by more than 50ms, and the poorest slowdown is less than 500ms, indicating a fine tail latency.

(18) **Comments**: *D14) Since WRL is a workload level statistic, I assume that each "epoch" in your plots are the entire query workload? If so, the train/test split strategy here is confusing. Are the query templates in the "trian" and "test" set distinct, or do you only split instances between the train and test set? As written, I assume the query templates are mixed, which makes the results quite a bit more confusing. For TPC-DS, almost all query instances have the same optimal plan, so this is essentially the same as having identical train and test sets.*

*JOB, however, is explicitly designed to have a different optimal plan for each query, so splitting by instance makes more sense, however in this case the optimizer *never* gets to train on a plan with an distinct optimal plan?*

An easier train/test split strategy might be "cross validation over time." If you look up Pari Negi, he has some cardinality estimation papers with the "JOB extended" queryset. This contains 1000+ JOB queries. Then you can train by never seeing the same query twice.

Response: Thanks for your advice. As queries of TPC-DS and Stack Overflow are automatically generated, we agree that queries of the same template have nearly identical optimal plans. Thus for TPC-DS and Stack Overflow dataset, we divide training and testing workload by templates in the revised version. We pick 5 templates of TPC-DS as testing workload, ensuring that all tables in testing workload exist in training workload. For Stack Overflow, we pick templates 1, 3, 5, 12, 14 and 16 for testing, and use the rest as training workload.

For JOB, we divide the training and testing dataset by instance to see LOGER's performance on all templates, which can also support other analyses including tail latency and improvements on each template. In addition, we use a random splitting strategy used by Balsa in the revised version. In both cases, the plans generated by LOGER perform best.

(19) **Comments**: *D15) LOGER has shockingly good first-iteration performance. It does not seem to have a "cold start" problem at all. A large portion of this seems to come from the left-deep restriction in F15, and the manual beam schedule which essentially forces LOGER towards the built in optimizer. Are these the only two reasons? Can you elaborate on this rather shocking result?*

Response: Thanks for your insightful comments. We conduct experiments in ablation study to show the components that contribute to alleviating the cold start problem. Please further check the feedback to item 3 of the meta-reviewer.

(20) **Comments**: *D16) The performance of LOGER on commercial system is also impressive. If you are reporting numbers in relative terms, you should just name the system. Hopefully it is a system with a powerful baseline optimizer, like SQL Server. If it is, this is probably of significant interest to Microsoft.*

Response: Thanks! The commercial DBMS is Oracle, and we perform tests on it with the default configuration parameters. We are still afraid that straightforwardly announcing the name of the commercial DBMS might result in some legal issues. Thus, we use CommDB instead of its name to represent it as other papers do, like JOB and Balsa.

We did not choose SQL Server, as its capability of query hints is not as powerful as PostgreSQL and Oracle. Since SQL server can only forbid specific operators at the scope of the entire query plan, we failed to implement ROSS on SQL server. In contrast, query hints that disable specific operators can be applied to each logical join node in PostgreSQL and Oracle.

(21) **Comments**: *D17) An experimental comparison to simpler methods should be made here. The restriction of the beam search, especially for small queries, is quite extreme. What if I used a robust left-deep ordering? Or disabled nested loop join? Establishing some simple baselines like this will help me better gauge the results.*

Response: We add a simple baseline that disables the merge join operator in PostgreSQL, named as PG-NoMerge. Please further check the feedback to item 6 of the meta-reviewer.

(22) **Comments**: *R1) Clean up the explanation of the loss function, and specify where each data point comes from.*

Response: We have simplified the loss function and improved the explanation in the revised version. Please check the feedback to D10.

(23) **Comments**: *R2) Explain and justify how data is divided between training and test set in the experiments, and explain what an "epoch" consists of.*

Response: Please further check the feedback to comment D14 above. We have made an explanation of the term *epoch* in the revised paper. In an epoch, we shuffle the queries in training workload, iteratively generate plans for each query, and train the networks by a mini-batch sampled from experience dataset after each iteration.

(24) **Comments**: *R3) Provide numbers in absolute terms in addition (or instead of) relative terms.*

Response: We have added corresponding figures of experimental results (Figure 9 in the paper) as your suggestion indicates.

(25) **Comments**: *R4) Compare with simple baselines, like disabling problematic operators or using a robust left-deep ordering.*

Response: We compare LOGER with PostgreSQL that disables the merge join operator as a baseline in the revised version. Please further check the feedback to item 6 of the meta-reviewer.

(26) **Comments**: *R5) Explain the fantastic avoidance of the cold start problem, and hopefully give readers some intuition behind the results!*
Response: Thanks for your kind comment! Please further check the feedback to item 3 of the meta-reviewer.

(27) **Comments**: *R6) Expand the experimental analysis to include some of the larger and more diverse datasets used for QO work in the past, like extended JOB or Stack.!*
Response: We have added a large dataset, Stack Overflow, and reported the results in experiments.

RESPONSE TO REVIEWER-3

(1) **Summary**: *The paper proposes LOGER, a learned query-optimizer that is designed for efficiency and robustness. LOGER uses graph transformers to capture relationships between tables and predicates. It then uses a beam search-based method to find promising join candidates with adaptive exploration. Lastly, LOGER uses a multi-task loss function with reward feedback transformation to enhance the model robustness. The experiments on JOB and TPC-DS benchmarks show that LOGER achieves an up to 2.05x query speedup compared to prior learned query optimizers.*

(2) **Comments**: *W1: LOGER uses one-hot encoding of the table in multiple feature representations, which seems to prevent the model from generalizing to updated schema and new workloads.*
Response: Thanks for your advice! Please further check the *Technical Extension* in the summary part.

(3) **Comments**: *W2: The only comparison against state-of-the-art methods is in Table 1. The paper should have a more comprehensive evaluation against prior arts, including the analysis of the training cost, inference time, and query performance across different workloads and dataset sizes.*
Response: We have performed corresponding experimental studies following your suggestions in the revised version. The extension includes the time consumption of model training and inference, the use of a large dataset Stack Overflow, more competitors including Bao and PostgreSQL with a disabled operator, and a deeper investigation of techniques that alleviate the cold-start problem.

(4) **Comments**: *W3: The authors should also compare LOGER against Bao, especially since LOGER shares similar insight with Bao by utilizing knowledge of traditional non-learning optimizer by restricted operator search space, as the authors mentioned.*
Response: Thanks for your advice! Bao is a representative work of learned query optimization and earned the best paper prize in SIGMOD 2021. Now, we have added the results of Bao in the experimental parts. Please further check the feedback to item 7 of the meta-reviewer.

(5) **Comments**: *D1. I agree that the query representation needs to be expressive. But I don't think the adjacency matrix does not contain the graph structure. It's also a different way to represent the graph structure.*
Response:. Thanks! We agree with your comments and modify our presentation in the revised version.

(6) **Comments**: *D2. Since the table representation contains a one-hot encoding, how would the model generalize to updated schemas or even different databases?*
Response: We have modified the model and removed the one-hot encoding following your advice. Now the revised LOGER can support schema updates. Please further check *Technical Extension* in the summary part.

(7) **Comments**: *D3. Does the pooling method in the query representation matter? Have the authors tried other pooling methods (e.g., average pooling) than max pooling?*
Response: We agree that the pooling method is an important hyperparameter, and we are also interested in whether other pooling methods work. From the experimental results, we find that max-pooling produces the best WRL and training stability but sum-pooling and mean-pooling yield a slightly better GMRL. We guess that through max-pooling, the query representation module pays more attention to columns with the smallest selectivities but tends to ignore information from other columns.

(8) **Comments**: *D4. It seems that the operator representation discussed in Sect 4.2, which concatenates one-hot vectors of each table, cannot generalize to new schemas/databases either.*
Response: We have redesigned the value model to solve the one-hot issue, which is explained in the response to W1. We replace the one-hot operator representation with the average of operator representations for all tables. Please further check the *Technical Extension* in the summary part.

(9) **Comments**: *D5. The authors may consider only showing a selective set of graphs for the ablation study (Figure 11 - Figure 15). Then, the authors can have more space for a comprehensive evaluation against state-of-the-art methods.*
Response: Thanks! In the revised version, we remove some of the figures to make room for substantial extensions to experiments and report more results. Please further check *Experiment* in the summary.