Thanks for your careful and valuable comments. We will explain your concerns point by point.

**Some high-frequency questions come from different reviewers.**

**Q1:** While PPO achieves a small improvement over the state-of-the-art approach Feedback, it is unable to get anywhere near OPT.

**A1:** OPT only optimizes the query latency metric. It knows precisely which queries will arrive, so it can pre-deploy the best partition schemes, regardless of the consequences of too many re-partitionings. Therefore, for these workloads (e.g., SY1500, lineitem, supplier) with strong periodic and few query types, PPO and OPT are close in latency (see Fig.13). For other low periodic workloads (e.g., SY4000, store\_sales, web\_sales) with diverse queries and random arrival times, PPO deploys a compromise partition scheme. It will not frequently repartition to deploy some low-value partitions, so PPO reduces a small amount of query latency (Fig.13) but saves a significant amount of repartition time (Fig.14), compared with OPT. To cover any scenario that might occur in reality, we generate 10 types of workloads. Some workloads may make PPO achieve a slight improvement, but in fact, PPO avoids deploying many inefficient partition schemes.

Considering that some workloads may not exist in reality, we plan to refer to some good ideas to improve their characteristics in the future. E.g., Ref [55] (the first paragraph of section3.2,page147) defines the future workload as follows: the incoming new query exists in the pre-collected buckets and then we change only the frequency of the old query to represent it.

**Q2:** The paper is very badly written.

**A2:** I am very sorry that my handwriting has caused a lot of misunderstandings. I only rely on some software, e.g., DeepL and Grammarly, to correct my handwriting. But my English writing is still too poor, so I will look for a professional translator to correct my mistakes and dictions.

**Reviewer#1:**

**D1:** Meaning of "feature" unclear.

**A1:** Here, it mainly refers to the filter conditions and projection columns of the OLAP-style workload.

**D3:** Limitations: meaning of "worthless" very unclear.

**A3:** These worthless queries can be understood as occurring infrequently or arriving at long intervals, and there is little value in designing partitions for them.

**D4:** "..clusters to approximate the workload trend in future period". Above, you just said that was impossible to predict, please make this consistent.

**A4:** My description is unclear. It should be revised as "...clusters to approximate the clustering characteristics of queries in the future period."

**D7:** Do you assume that you KNOW the I/O and repartition costs accurately?

**A7:** No, the offline training of RL is assisted by the cost model. Since the partitioning does not support the what-if mode in the database, it is expensive to get real metrics as training data.

**D10.** Fig 2 does not show how old partitions are re-partioned and it seems that the news one are inside the database while the old ones are not, which is inaccurate.

**D17:** "the old partition scheme in finer granularity". Very unclear what that means

**A10/17:** I'm sorry that Fig2 does not show the process in detail. Some relevant descriptions about "How to replace old partitions"” can be seen in Section 7. The overall meaning of "finer granularity" is that many old partitions may not be completely replaced, which depends on the degree of conflict it has with the new partitions, and the query clusters obtained by the selector further eliminates some of the conflict (Algo 2).

**D12:** Does your cost model consider materialized views? It should, to be realistic.

**A12:** Current HDD model does consider the effect of materialized views, But the materialized view selection technique (Ref [a]) can help partitioning optimize some queries with Group By, Aggregate, and Join.

[a]Yuan, Haitao et al. “Automatic View Generation with Deep Learning and Reinforcement Learning.” ICDE2020.

**D14:** Where do the 1.5 values come from?

**A14:** The value 1.5 comes from the box plot method and is used to compute outliers' upper and lower limits.

**D16:** If you don’t use an existing clustering algorithm, where do you prove that the results are the true, actual clusters?

**D19:** please argue for the correctness of alg 1

**A16&19:** We plan to add an extended experiment for a new baseline (PPO without Selector) and changes to the queries that participate in repartitioning when the selector is or is not used.

**D21:** "The initial state is all tables are not partitioned" Very unclear what this means.

**A21:** It indicates that the table does not apply any partition optimization, and all blocks are organized according to the default order of the tuples.

**D22:** Unclear exactly what a repartition operation does, give concrete detailed example(s), e.g., what happens to old data, do you treat old data differently than new data, since new queries may only touch new data, etc.

**A22:** This paper considers only analytical queries for now, so the data is considered stable and We will not consider new data for now.

We give a simple example to illustrate how old partitions are replaced. Given two old partitions {P1(a1,a2,a3), P2(a4)}, and P1 store 3 blocks {B1、B2、B3} while P2 store 2 blocks {B4、B5}. At the next repartition point, the system is asked to deploy the new partition P3(a3,a4) to replace P2. Then we fetch these mini-pages storing attribute a3 from B\_{1,2,3} and write them into B\_{4,5}.

**D27:** "Hard disk cost model" Today, most storage is on SSDs, how does that influence your design ?

**A27:** The RL is independent of the HDD, so when selecting a new cost model for SSD, we only need to retrain and fine-tune the agent.

**D28:** Any new contribution related to SCVP ? Seems not.

**A28:** Many static vertical partitioning technologies are applied for different scenarios, e.g., memory database (Ref 33) and Spark+HDFS (Ref 22). So, we separate the PPO model from static VPs to enhance its scalability.

**D29:** what is the cost of retraining ?

**A29:** The GPU resource and training time.

**D32.** Cant you use native PostgreSQL partitions instead of tables?

**A32:** PG only supports hash, range, and list partitioning. The only platforms supporting custom partitions are Spark SQL and Vertica.

**D36(1).** baselines are 6-11 year old, not SOTA

**A36(1).** Sorry, I have not found any other SOTA yet. Recent studies on vertical partitioning mainly focus on static partitioning (Paper [a]). The studies on dynamic partitioning are to select the primary/foreign keys used for establishing the hash partitions (paper[b] and Ref 55).

[a] Campero Durand G, Piriyev R, Pinnecke M, et al. Automated vertical partitioning with deep reinforcement learning. ADBIS, 2019.

[b] Parchas, Panos et al. “Fast and effective distribution-key recommendation for amazon redshift.” VLDB, 2020.

**D36(2):** 5 HOURS to train for such tiny tables, this will not scale to realistic data sizes

**A36(2):** The offline training time of RL depends mainly on the HDD and the estimated repartition cost. It is not affected by the actual table size.

**D42:** [26] and [33] should be used as experimental baselines

**A42:** The metrics of Ref 33 will be added in later work. However, Ref 26 doesn't share the code, and we have yet to receive a response to our previous inquiry email.

**D2 /D5-6 /D8-9 /D11-13 /D15 /D18 /D20 /D23-26 /D30 /D34 /D35 /D37-38 /D43.**

**A:** Thanks for your advice. We will revise these problems.

**Reviewer#2:**

**W1(1):** Given that this is an ML-based approach, wouldn't it make sense to predict this benefit?

**A1(1):** Like cardinality/cost estimation, I have also tried to train an n-LSTM model to learn query plans and partitions to fit the real execution time. But the training data is expensive because the DB platforms do not provide a what-if mode for partitioning (unlike the index or view), i.e., partitions must be deployed on the DB to get an estimation for the optimizer.

**W1(2):** In Fig.11, the PPO approach essentially learns how to imitate the HDD. We can also see that the OPT does not.

**A1(2):** To achieve the lowest query latency, OPT does not consider the rapid increase of repartitioning cost. Perhaps removing the OPT in Figures 11 and 14 would make the figures easier to understand.

**W1(3):** A truncated y-axis.

**A1(3):** The reason is we focus on explaining that PPO improves the Feedback to be closer to OPT, rather than showing how much it reduces query latency for Feedback. If the difference between OPT and Feedback is small, then the query latency reduced by PPO must also be small, which means there is little improvement after applying dynamic partitioning.

**W2:** The authors do not explain their choice of the HDD Cost Model. There are several cost models in the literature that could have been used to estimate the cost/benefit-ratio of repartitioning. In fact, at EDBT 2022, there was a paper that proposed a partition advisor. How is the HDD Cost Model superior or different from these already existing cost models?

**A2:** Our choice is based on the correlation between the estimated cost and the actual execution time of PG. Many cost models have specific limitations, such as the in-memory (Ref [21]) or distributed environment (Ref [22]), horizontal partitions (Ref [a]), and low positive correlation (Ref [20]). We find that paper [a] has some designs that can be referenced to improve HDD, e.g., cold/hot partitions, uncompressed/compressed column partition size.

[a]Brendle, Michael et al. “SAHARA: Memory Footprint Reduction of Cloud Databases with Automated Table Partitioning.” EDBT (2022).

**W4:** The paper is very badly written and would need to undergo substantial rewriting in order to be publishable. Finally, the TPC-benchmarks are also synthetic benchmarks.

**A4:**Thanks for the correction. I plan to find new real datasets, such as OSM and IMDB.

**Reviewer#4:**

**D2:** First of all, the focus seems on read-only workloads, and this is not realistic. Also in the experimental setting I could not figure out exactly which queries were used.

**A2:** I am sorry that the experiments are currently positioned in a read-only scenario. The query types consist of the 22+99 queries provided by the TPC. They are standardized, like the repository (https://github.com/palatinuse/database-vertical-partitioning/blob/master/src/db/schema/BenchmarkWorkloads.java). These workloads (.csv) are available in the "/data" folder of our repository.

**D3:** Also, the paper is written such that it is not clear that the reinforcement learning is adopted only for the decision of "when" to run the partitioning, and not how.

**A3:** Since there are many partitioning techniques (Ref 17-23) for different platforms, such as traditional databases (standalone), distributed and in-memory databases, etc. So, this paper treats them as black boxes. So, we treat them as black boxes and discuss some other important details of "how to partition". For example, we search and format historical queries to generate partitions (Sec 5.1), including clustering, filtering, and outlier detection, which directly affects the potential reward of new partitions. We also determine how to update conflicting old partitions when faced with new ones (Algo 2).

**D4:** A lot of technical details are obscure and require a lot of guesswork to be derived from the text. For instance, it is said that the clustering is not done via K-means or DBscan, but it is not spelled out in clear terms what is the input, what is the output, and what is the exact algorithm of the clustering performed here.

**A4:** Thanks for the correction. The inputs of our clustering strategy are a repartitioning time and the historical queries arriving at each time point (Algo 1: line 5). The output is a collection of clusters whose attributes is non-overlapping each other. Assigning queries to different clusters is done in two steps: construct (Algo 1: lines 6-10) and merge (Algo 1: lines 11-20) clusters.Wherethe similarity between queries is computed by their accessed columns, and when the clusters are completed depends on 𝛼.

**D5:** Many details of the experimental settings are also obscure. The gain reported is the sum on a long workload, but this is just one measure. The actual gain per query seems negligible and data is not reported on best/worst/median gain incurred by this method. I am also surprised that the basic running time where all columns are separate, the column store approach, is not presented as baseline.

**A5:** Only Fig.10 reports the reduction in average query cost for every workload. All the results in the paper were obtained by repeating the experiment three times and taking the average value. Showing the best/worst/median of the results is a good suggestion, and we will improve it in the future version.

The default table is set to column storage, i.e., each column corresponds to a vertical partition. This method is not shown due to its higher (sort) merge cost. But it is still an excellent reference to highlight the advantages of dynamic partitioning, and we will add it in the new version.

**D6:** It is written that the queries are manually mapped to the tables. This is strange. Also it is reported that steps have been takes to avoid the intervention of the optimizer, and this is strange as well. Finally, it seems that non primary indexes are not used. This is also quite odd. There are other technical issues that are unclear, for example note that postgres does not support clustered indexes, so I am confused by the fact of that type of features.

**A6:** Sorry, our description of "manual" is wrong. We still use the optimizer and just replace the routing query functionality of it. For a query involving multiple partitions, it will be automatically routed into multiple sub-tables by the program. Besides, the primary index has been built on an additional PK column in each partition.

HDD requires indexing information(Ref 23). We have not explored the influence of indexes. The main reason is that when considering the index, the estimated cost is very different from the actual execution cost in PG (using hash indexes). So, it is necessary to improve the HDD before conducting the index experiment.

**D7:** There are many grammatical issues, abstract and introduction is filled with errors in the usage of plural&singular words.

**D8:** the readme misses command to run the experiments, instructions to change datasets/ queries.

**A7/8:** Thanks for your advice. I will revise it carefully.

**Reviewer#5:**

**D2:** It is not clear why experiments for the two TPC datasets are per-table, as the authors also state in the paper that queries typically involve multiple tables.

**A2:** The HDD (Ref 23) does not support the cost estimation of the join operations (e.g., hash join). We have preprocessed the multiple-table TPC queries into single-table queries. This repository (<https://github.com/palatinuse/database-vertical-partitioning/blob/master/src/db/schema/BenchmarkWorkloads.java>) gives the split queries from Q1~Q22.

**D3:** Similarly, it is stated rather briefly how the created partitions are used in PostgreSQL. Two options are discussed and, a bit surprisingly, the option that is not using the PGs query optimizer is used. This perhaps requires a bit more explanations. How is this "manually" done for complex queries, e.g., the ones in TPC-H?

**A3:** I'm sorry that our description of "manual" is wrong and it has caused many misunderstands. We just replace the routing query functionality from the optimizer, not without it.

Take TPCH Q4 as an example to explain how it routes to different partitions. First, It will be split into two queries: (1) Q4\_1-> select o\_orderpriority, count(\*) as order\_count from orders where o\_orderdate >= date '1996-03-01' and o\_orderdate < date '1996-03-01' + interval '3' month group by o\_orderpriority order by o\_orderpriority LIMIT 1; (2) Q4\_2-> select \* from lineitem where l\_commitdate < l\_receiptdate. Second, taking Q4\_2 as an example, suppose our router distributes Q4\_2 to two partitions: P1[PK, l\_commitdate, l\_receiptdate], P2[PK, other columns]). We build a hash table for every PK. Finally, we run the nested query "select \* from $P2 where PK in (select PK,l\_commitdate,l\_receiptdate from $P1)". All results are (sorted and) merged.

**D4:** The authors state that existing solutions do not do well in finding out which queries to consider to decide how to partition. But it seems that the authors’ approach is not free of tuning parameters that require specific knowledge or a behavior of future queries the approach can learn at all (e.g., in Section 5.1).

**A4:** In fact, these parameters are computed in an unsupervised manner. The 𝛼 is computed by the density of historical queries because it determines when each cluster ends (affinity in {time, attributes}=>density). 𝛽1/𝛽2 are determined based on the sizes of collected clusters because they need to filter these too-large or too-small clusters (outlier detection =>box-plot).

Tuning RL has a lot of tricks. E.g., we can use the “warm-up” strategy to tune the learning rate. In the early stage of training, we set the learning rate to a smaller value to make the RL learn various knowledge and gradually increase its value in the late stage so that the RL can converge faster.

**D5:** It is unclear how the numbers in Figure 12 are related to runtimes in Figure 13.

**A5:** In Fig.12, the y-axis represents the sum of the query latency for these queries arriving at the corresponding period (x-axis). Fig.13 only shows the total query latency for the entire workload.