Short Paper

Query Performance Explanation through Large Language Model for HTAP Systems

Haibo Xiu*
Duke University
Durham, NC, USA
haibo.xiu@duke.edu

Li Zhang
ByteDance US Infrastructure System Lab, ByteDance, Inc.
San Jose, CA, USA
li.zhang@bytedance.com

Tieying Zhang
ByteDance US Infrastructure System Lab, ByteDance, Inc.
San Jose, CA, USA
tieying.zhang@bytedance.com

Jun Yang
Duke University
Durham, NC, USA
junyang@cs.duke.edu

Jianjun Chen
ByteDance US Infrastructure System Lab, ByteDance, Inc.
San Jose, CA, USA
jianjun.chen@bytedance.com

Abstract

In hybrid transactional and analytical processing (HTAP) systems, users often struggle to understand why query plans from one engine (OLAP or OLTP) perform significantly slower than those from another. Although optimizers provide plan details via the `EXPLAIN` function, these explanations are frequently too technical for non-experts and offer limited insights into performance differences across engines. To address this, we propose a novel framework that leverages large language models (LLMs) to explain query performance in HTAP systems. Built on Retrieval-Augmented Generation (RAG), our framework constructs a knowledge base that stores historical query executions and expert-curated explanations. To enable efficient retrieval of relevant knowledge, query plans are embedded using a lightweight tree-CNN classifier. This augmentation allows the LLM to generate clear, context-aware explanations of performance differences between engines. Our approach demonstrates the potential of LLMs in hybrid engine systems, paving the way for further advancements in database optimization and user support.

Keywords

Query Optimization, Retrieval-Augmented Generation (RAG), Large Language Model (LLM)

1 Introduction

"Why does my query run so slowly?" In modern database management systems (DBMS), users frequently struggle to understand why certain queries experience very long execution times. While contemporary optimizers provide an `EXPLAIN` function that details the execution plan, these explanations are too complex for non-experts to interpret fully. This challenge is particularly evident in hybrid transactional and analytical processing (HTAP) systems, such as ByteHTAP [4], which features a unified interface with two underlying execution engines: OLTP (online transactional processing, referred to as TP) and OLAP (analytical processing, referred to as AP). When a query is executed in the HTAP system, users often seek guidance on why one engine outperforms the other. Traditionally, database experts manually analyze queries to provide tailored explanations, but as query volumes grow, this approach becomes unsustainable.

To fill the gap between the incomprehensibility of optimizer-generated explanations and the high cost of expert-provided explanations, we aim to develop a user-friendly and intelligent explanation system satisfying three key criteria:

- The explanations should be clear and easy for non-experts to understand why certain engine is faster than the other at runtime.
- The explanations must consider the database and engine context to ensure reasonably accurate output, even if they may not always achieve the same level of precision as human experts.
- The system should support fully automatic generation with minimal human effort, while remaining cost-efficient in both training and maintenance.

To meet these requirements, we propose a novel framework that leverages large language models (LLMs) to automatically explain query performance across different engines and help users understand the reasons behind performance differences. LLMs have gained popularity thanks to their ability to generate understandable natural language outputs. However, balancing accuracy and efficiency presents a trade-off: while using pre-trained models (such as Doubao [3], ChatGPT [1], Llama [15], and Claude [2]) is efficient, they lack the specific context needed for accurate query performance explanations. Although fine-tuning LLMs could improve relevance and accuracy, this is resource-intensive. To strike a balance between cost and accuracy, we employ pre-trained public models using a Retrieval-Augmented Generation (RAG) approach [9], addressing the limitations of general-purpose LLMs while maintaining efficiency.

The RAG framework relies on two key components: a retriever and a knowledge base [7]. First, the **retriever** finds relevant references in a pre-built knowledge base and provide them as contextual input to the LLM. In the HTAP system ByteHTAP [5] that this paper focuses on, this retriever is powered by a lightweight machine learning model—specifically, a tree-CNN classifier based on recent research on learned query optimizers [11, 17, 20], trained to route queries to the engine best suited for efficient execution. This model also functions as a query plan encoder, generating embeddings that represent the plan for each query.

Second, the **knowledge base** serves as an external data source, supplementing the LLM's original training data. In our setup, the knowledge base stores historical queries, their plan embeddings

*This work was done during a summer internship at ByteDance US Infrastructure System Lab, San Jose, CA, USA.

EDBT '26, Tampere (Finland)

generated by the tree-CNN classifier, and expert-curated performance difference explanations—specifically, the reasons why the TP plan runs faster than the AP plan and vice versa.[1] At runtime, the retriever searches the knowledge base for similar plan embeddings relevant to the current query. These retrieved embeddings enrich the LLM with relevant context-specific information, enabling it to generate more accurate and targeted natural language explanations. As shown by experiments, by combining the precision of the machine learning model's embeddings with the LLM's generative capabilities, our approach provides insightful, contextually grounded explanations of engine performance, without requiring expert intervention or costly fine-tuning of the LLM. Our framework is not limited to ByteHTAP; it is adaptable to any HTAP system capable of retrieving historical explanations as augmented inputs.

Explaining why one engine outperforms another may seem more straightforward than answering the broader question of "Why does my query run so slowly?" However, the two are intricately connected, as engine choice and query execution performance are shaped by the same underlying factors, such as plan efficiency and system architecture. While fully automating explanations for query performance remains challenging, our approach offers a significant step forward. By integrating LLMs within HTAP systems, our framework helps users understand engine performance differences, demonstrates the potential of LLMs to deliver intuitive, accessible explanations, and paves way for future research toward more comprehensive automation in database performance analysis.

The structure of this paper is as follows: After briefly surveying related work in Section 2, we describe our framework in Section 3, including the integration of LLMs with RAG to improve explanation quality in HTAP systems. Section 4 and Section 5 outline the knowledge base construction and prompt engineering, while Section 6 presents and analyzes results of experiments. Finally, Section 7 concludes with key insights and future research directions.

2 Related work

Large Language Models (LLMs) have shown considerable promise in enhancing database systems by providing intuitive, natural language explanations and recommendations [19]. LLMs excel at translating complex, system-level insights into user-friendly explanations, sparking interest in their application across various database tasks, including query performance analysis and optimization. LLMs have also been effectively applied in database research. For example, they have been used to understand and generate SQL queries [6, 12, 13, 16]. D-Bot [18] employs LLMs to detect and resolve anomalies within databases, while Panda [14] leverages Retrieval-Augmented Generation (RAG) [9] to ground LLM outputs in context, providing performance diagnostics based on execution metrics rather than query plan analysis. RAG improves the accuracy and relevance of language models by integrating a retrieval mechanism with text generation [7]. Further research, such as DBG-PT [8], demonstrates the utility of LLMs for diagnosing performance regressions through comparisons between structured query plans. While DBG-PT successfully analyzes plans from the same optimizer, our work extends this approach to scenarios involving plans generated by different engines. Instead of comparing the plan details from the `EXPLAIN` clause directly, our methods enhances explanation quality by integrating RAG by a small model for improved context and relevance.

[1]These explanations address why one engine's plan runs faster than the other post-execution, rather than interpreting the classifier's routing decision—the latter underscores a broader challenge of interpretability in machine learning.

3 Retrieval-Augmented Explanation Generation by Large Language Model

3.1 Constructing an Effective Retriever

Since LLMs may lack query-specific or up-to-date context, we leverage insights from past queries through RAG to enrich responses with precise and relevant historical information. These retrieved references, combined with carefully designed prompts, are provided to the LLM to generate more accurate and contextually grounded answers.

Knowledge base for RAG. In our system, we store historical query plans and their corresponding plan performance explanations in a key-value knowledge base. The key consists of a pair of plans (one for TP and the other for AP) for the same query, while the value contains the plan details and associated explanations. Instead of straightforwardly storing the plan pairs as raw text (e.g., the output of `EXPLAIN` from the optimizer), the plan pairs in our system are stored as vectors, whose encoding process is described later. This idea is motivated by our focus not on semantic similarity between plans but on "similar performance distinctions"—specifically, similar performance differences between TP and AP plans observed in past queries. For a new query, we aim to retrieve historical queries with similar plan performance distinctions and use this knowledge to enhance the LLM's generation.

Plan embeddings by a lightweight model. Our HTAP system [5] features a smart router, which is an enhanced tree-CNN classifier that predicts, for a given query, which engine will yield a plan with better performance. Experiments demonstrated that the router achieves high accuracy in identifying the more efficient plan between TP and AP engines. Therefore, it naturally serves as a good model for generating plan embeddings. Key advantages of using the smart router for plan embeddings include:

- *Lightweight model:* The smart router is highly efficient, with a physical model size of less than 1MB and an average inference time of only 1ms, making it an ideal choice for embedding generation. Additionally, it can be quickly retrained to adjust to changes in query workloads or underlying data.
- *Task-specific design:* Directly taking plan trees as input, embeddings generated by the smart router can capture detailed plan performance comparisons since the original task is to determine the faster engine. Trained on a large dataset of query plan pairs, it identifies performance-relevant features within plans. These intermediate plan encodings serve as "signatures" enabling the retriever to match new queries with similar historical performance distinctions.

3.2 Framework Overview

From a system integration perspective, our explainer operates above the TP/AP optimizer in the ByteHTAP system. The system steers a pre-trained LLM to generate explanations based on plan embeddings from the smart router and retrieved contextual information. The framework consists of three main components as illustrated in Figure 1. We describe these components below in turn.

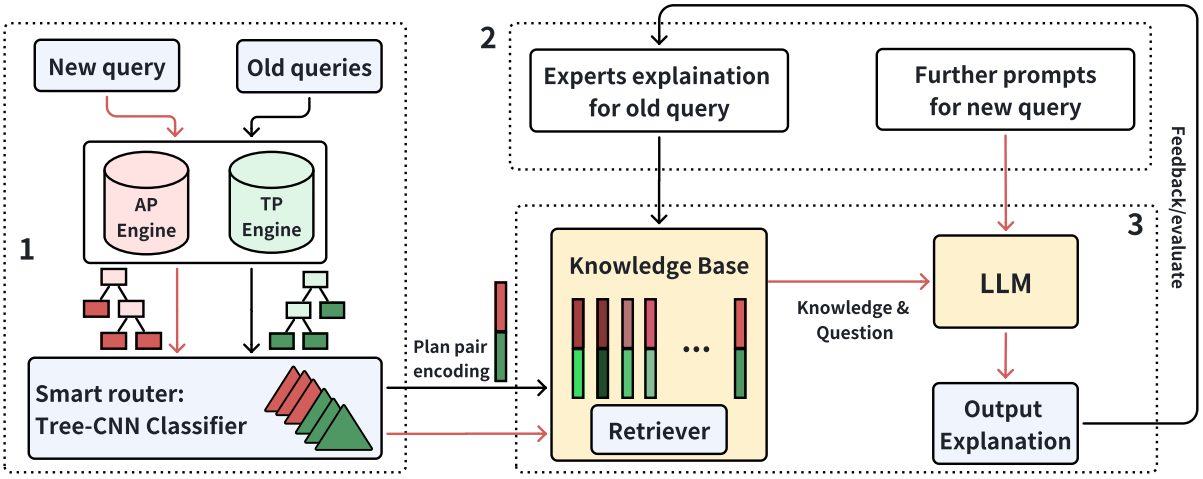

Figure 1: Framework of our method with three main components: 1) HTAP system, 2) human (user/expert), and 3): RAG and LLM. The red line shows the workflow for new queries, while the black line represents historical queries.

ByteHTAP system with smart router. This component is marked as **1** in Figure 1. To explain the performance distinction between engines for a new query (marked by red arrows), the tree-structured execution plans from both AP and TP engines are processed by the smart router, which encodes them into a vector representation. The plan pair embedding, created by concatenating vectors from both AP and TP plans, is then utilized by subsequent components to generate explanations. Historical queries (marked by black arrows) are selected from the training set of smart router and forwarded to the human side for expert-curated explanations, which are then stored in the knowledge base along with their corresponding plan pair embeddings.

Human interaction and expert evaluation. The human side (marked as **2**) has two roles: database users and database experts. Users submit queries and seek guidance on performance-related questions. Experts can provide, during the knowledge base construction phrase, detailed explanations of why one plan performs better or worse than the other based on practical insights. They can also assess the quality of future LLM-generated explanations. Users may also offer additional contextual information, such as details on newly created indexes, which help refine the LLM's responses and improve explanation accuracy.

RAG and LLM integration. On this side (marked as **3** in Figure 1), we integrate a knowledge base for retrieval and a pre-trained public LLM for explanation generation. The knowledge base is a vector database populated with historical queries, where their AP/TP execution plans are encoded by the smart router. The resulting plan pair embeddings are stored as keys and the values are the expert's explanations. Further details on the construction of the knowledge base are provided in Section 4. For each incoming user query, the retriever uses the plan pair embedding obtained from the smart router to search in the knowledge base for the top K most similar plan pairs. These retrieved knowledge (including expert explanations), the plan details and execution results of the new user query, and the background context (including default prompts and any user-provided prompts), serve as the input for the LLM.[2] The LLM then generates an explanation based on this enriched input, which is returned to the user.[3] As mentioned, these generated outputs can also be reviewed and evaluated by experts. If an explanation is deemed inaccurate, experts will correct it and add the revised version to the knowledge base for future retrieval.

4 RAG Knowledge Base Construction

For RAG, we construct a knowledge base by storing historical queries and their performance explanations. This knowledge base provides the necessary context for the LLM to generate accurate and relevant explanations. For each query, we store the following data: ⟨*plan pair encoding*, *plan details*, *execution result*, *expert explanation*⟩. The plan pair encoding is a vectorized representation of the pair of AP and TP plans, encoded by the smart router, which enables efficient retrieval of similar queries. Plan details includes the actual execution plans for both engines. The execution result indicates which engine executes this query faster. Finally, the expert explanation is a curated explanation from database experts detailing why one engine outperforms the other in specific cases.

To maximize the effectiveness of RAG, the knowledge base must include a sufficient number of representative queries. In a real-world scenario, this requires carefully identifying query patterns that frequently cause performance confusion for users. In our setup, we select query patterns frequently requested by ByteDance users seeking explanations for the engine performance discrepancy. Then, to protect the privacy of users' data and queries, we synthetically generate similar queries using the TPC-H dataset instead of directly using user-submitted queries. Note that these generated queries are also in the training set of the smart router, ensuring the encodings are attended to the performance distinctions. For the demonstration case in Section 6, the selected query patterns primarily include:

[2]Since the smart router may mispredict, the actual execution results may differ from the router's initial prediction. We include the execution results of the AP and TP plans to ensure that the LLM generates explanations based on accurate runtime information.
[3]If the LLM determines the augmented knowledge lacks sufficient information, it will return a None response.

(1) *Join queries*: Join operations in which AP and TP engines apply different join strategies, offering insights into engine-specific optimizations. These join queries vary in factors such as the number of joined tables, table size, predicate selectivity, and index usage.
(2) *Top-N queries*: Queries that retrieve the top N records based on specific criteria, often using clauses like `ORDER BY`, `LIMIT`, and sometimes `OFFSET`. These queries are common in user workloads and often perform differently depending on engine-specific optimizations.

After executing these synthetic queries on both TP and AP engines, we send the queries, plans, and execution results to database experts, requesting the corresponding explanations. Such expert participation is only required at the knowledge base construction time, and not needed in the subsequent explanation generation process. For the experimental setup in Section 6, we selectively include only 20 representative queries in the knowledge base. These queries are carefully hand-picked and designed to reflect the types of performance difference issues most frequently encountered by online users. While 20 queries alone may not cover the full spectrum of all possible queries, they are sufficient to capture the core patterns behind the cases users are most interested in. This selective approach not only helps reduce the cost of expert annotations but also ensures that the knowledge base is able to capture the performance distinctions that are broadly applicable to the users' query workload. In addition, we provide an interface to continuously expand the knowledge base by accepting new queries along with expert-provided explanations, allowing the system to evolve and improve over time. A full analysis of how to manage the knowledge base, including methods for automatically selecting representative queries and expiring stale queries, remains a future work direction.

For each query, the plan pair encoding is a 16-dimensional vector generated by the smart router. The retriever searches the top 2 similar vectors for the new query. To evaluate generalization and test our performance, we synthesize an additional set of 200 queries, distinct from both the training and representative query sets, following the same distribution of query patterns we consider. Detailed experimental results are in Section 6.2.

5 Prompt Engineering

Following the RAG process, we send the retrieved information along with the new query to the LLM. To guide the LLM in generating accurate explanations, we provide carefully structured prompts, which are organized into three parts:

- *Background information*, which includes the overall objective, an overview of the HTAP system, key differences between its engines, and specifics about the schema and dataset queried.
- *Task description*, which defines the RAG task, detailing expected inputs and outputs, along with additional guidance to ensure clarity.
- *Additional user-provided context*, such as recent modifications to indexes, to ensure that the LLM has the latest context.

During prompt design, we observed that the pre-trained LLM often defaults to directly comparing the plan costs generated by the query optimizer to explain which plan is faster. However, because the optimizers are implemented differently across engines, plan costs are generally not comparable across engines (which is the case for ByteHTAP). To prevent this incorrect reasoning, we emphasized in the prompts that the costs in the plan pair should not be used for comparison. Table 1 presents the prompts used in our experiments, and Section 6 analyzes the generation results.

Background information: *We are using RAG to assist database users in understanding query performance across differences engines in our HTAP system—specifically, why one engine performs faster while the other is slower. Please ensure you are familiar with the TPC-H schema, and our dataset follows the default schema and contains 100GB of data. Our HTAP system has two database engines, "TP" and "AP". The TP engine uses row-oriented storage, while the AP engine utilizes column-oriented storage. Note that the optimizers for TP and AP engines are distinct, leading to different execution plans. Therefore, you are not allowed to compare the cost estimates of the execution plans from TP and AP engines.*
Task description: *Here is your task: I will provide you with the execution plans and their performance results from both the TP and AP engines. Please evaluate the performance of each engine without directly comparing the cost estimates. Focus on factors such as the join methods used, the storage formats (row-oriented vs. column-oriented), index utilization, and any potential implications of the execution plan characteristics on query performance. Your task is to explain the reason why the specific engine performs better for this specific query, based on these factors. To assist you, we have a retriever that can find relevant historical plans from the knowledge base with precise performance explanation from our experts. The* ***KNOWLEDGE*** *and* ***QUESTIONS*** *you received will be in the following format:* • ***KNOWLEDGE****: historical query + historical plan pair (AP/TP plan) + historical execution result (indicating whether TP or AP runs this query faster) + historical expert explanation (why TP or AP is faster).* • ***QUESTION****: new query + new plan pair + new runtime execution result.* *You could use KNOWLEDGE to explain the following new pair of plans in QUESTION. If the KNOWLEDGE does not contain the facts to answer the QUESTION return None. Note, to make sure your answer is accurate, I may input you several retrieved old queries with their plans, results and explanations. Please understand all the information I provide to generate your explanation. Now, I am ready to send you the KNOWLEDGE and QUESTION*
Additional user context: *Beyond the default indexes on primary and foreign keys, an additional index has been created on the* `c_phone` *column in the* `customer` *table.*

Table 1: Prompts used.

6 Experiments and Participant Study

6.1 Demonstrative Case

First, we show one example to demonstrate how our system works. This query is synthetic, constructed from similar query patterns we observed from the real user queries. All subsequent queries were executed in the same environment described in [5], consisting of a six-machine cluster with four data servers. Each data server is configured with 8 vCPUs, 32 GB DRAM, and 1 NUMA node.

Example 1. *Consider a query joining 3 tables.*

```
SELECT COUNT(*) FROM customer, nation, orders
WHERE SUBSTRING(c_phone, 1, 2) IN ('20', '40', '22', '30',
      '39', '42', '21')
AND c_mktsegment = 'machinery'
AND n_name = 'egypt' AND o_orderstatus = 'p'
AND o_custkey = c_custkey
AND n_nationkey = c_nationkey;
```

In this example, the TP plan takes 5.8s to run, while the AP plan completes in 310ms. We show the details of TP and AP plans in Table 2 and the corresponding expert's explanation and the LLM-generated explanation in Table 3. The explanation generated by our approach using the LLM demonstrates high accuracy. It highlights the key factor that hash joins are more efficient than nested-loop joins, as no index is available, which aligns closely with the expert explanation. The LLM-generated explanation also provides additional insights, including details about AP's aggregation efficiency, an aspect the experts did not explicitly mention. Overall, the LLM output is informative, clear, and easy to understand by non-experts. Because ease of understanding is a subjective measure, we conducted a user study to gather feedback

Details of TP Plan for Example 1
{ '*Node Type*': 'Group aggregate', '*Total Cost*': 5213.0, '*Plan Rows*': 1, '*Plans*': [{ '*Node Type*': 'Nested loop inner join', '*Total Cost*': 5175.0, '*Plan Rows*': 379, '*Plans*': [{ '*Node Type*': 'Nested loop inner join', '*Total Cost*': 1002.0, '*Plan Rows*': 285, '*Plans*': [{ '*Node Type*': 'Filter', '*Total Cost*': 2.75, '*Plan Rows*': 2, '*Plans*': [{ '*Node Type*': 'Table Scan', '*Relation Name*': 'nation', '*Total Cost*': 2.75, '*Plan Rows*': 25 }] }, { '*Node Type*': 'Filter', '*Total Cost*': 290.0, '*Plan Rows*': 114, '*Plans*': [{ '*Node Type*': 'Table Scan', '*Relation Name*': 'customer', '*Total Cost*': 290.0, '*Plan Rows*': 1142 }] }] }, { '*Node Type*': 'Filter', '*Total Cost*': 13.3, '*Plan Rows*': 1, '*Plans*': [{ '*Node Type*': 'Table Scan', '*Relation Name*': 'orders', '*Total Cost*': 13.3, '*Plan Rows*': 13 }] }] }] }
Details of Ap plan for Example 1
{ '*Node Type*': 'Aggregate', '*Total Cost*': 16500000.0, '*Plan Rows*': 1, '*Plans*': [{ '*Node Type*': 'Inner hash join', '*Total Cost*': 16500000.0, '*Plan Rows*': 134933, '*Plans*': [{ '*Node Type*': 'Filter', '*Total Cost*': 13500000.0, '*Plan Rows*': 13500000, '*Plans*': [{ '*Node Type*': 'Table Scan', '*Relation Name*': 'orders', '*Total Cost*': 0.5, '*Plan Rows*': 135000000 }] }, { '*Node Type*': 'Hash', '*Plans*': [{ '*Node Type*': 'Inner hash join', '*Total Cost*': 1640000.0, '*Plan Rows*': 135985, '*Plans*': [{ '*Node Type*': 'Filter', '*Total Cost*': 1500000.0, '*Plan Rows*': 1360000, '*Plans*': [{ '*Node Type*': 'Table Scan', '*Relation Name*': 'customer', '*Total Cost*': 0.5, '*Plan Rows*': 13600000 }] }, { '*Node Type*': 'Hash', '*Plans*': [{ '*Node Type*': 'Filter', '*Total Cost*': 3.0, '*Plan Rows*': 2, '*Plans*': [{ '*Node Type*': 'Table Scan', '*Relation Name*': 'nation', '*Total Cost*': 0.5, '*Plan Rows*': 25 }] }] }] }] }] }] }

Table 2: TP and Ap plan for Example 1.

Explanation by experts for Example 1
AP is faster than TP because TP's optimizer chooses to use nested loop join with no index available. AP uses hash join, which is more efficient.
Explanation by our approach for Example 1
AP is faster due to its use of hash joins and hash aggregates, which are highly efficient for handling large datasets, especially in a columnar storage format. These operations allow AP to scan only relevant columns and apply filters before joining, making it particularly effective for large, complex queries involving multiple tables. In contrast, TP's use of nested loop joins and group aggregates, combined with table scans that don't benefit from index optimizations, leads to slower performance when processing large datasets. AP's ability to efficiently process joins and aggregates across multiple tables gives it a significant advantage in this scenario.
Explanation by DBG-PT for Example 1
The AP engine is faster in this case because it utilizes column-oriented storage, which efficiently scans large tables like orders by only reading the required columns. Additionally, the AP engine uses hash joins, which are well-suited for joining large datasets. Both engines likely benefit from the index on `c_phone`*; however, the AP engine's column-oriented storage allows it to quickly access and filter the* `c_phone` *column without the overhead of retrieving entire rows. In contrast, the TP engine performs nested loop joins that are less efficient for large tables and requires reading entire rows, making table scans more costly. These factors combined result in the AP engine delivering better performance for this query.*

Table 3: Plan performance explanations for Example 1 by different approaches.

on how well users comprehended the generated explanations. The design and results of the user study are detailed in Section 6.3.

6.2 Experiment Results and Analysis

Explanation Accuracy. We first focus on experimentally evaluating the accuracy of the LLM-generated explanations. Given the lack of a standard evaluation method or benchmark for this specific setting, we relied on three database experts on HTAP systems to manually assess the generated explanations for correctness and completeness. The experts' evaluations revealed that in 91% of cases, the LLM-generated explanations were accurate and informative, though the explanations of the remaining 9% were less precise than expert interpretations, including 4% queries with `None` as output. The assessments showed high agreement among the experts, with only minor discrepancies in edge cases, which were resolved through discussion.[4] For incorrect cases, experts can correct the explanations and incorporate them into the knowledge base, improving accuracy for future retrievals.

Impact of retrieved vectors. To understand the impact of the number of similar vectors retrieved for augmented generation, we vary this parameter in the range from 1 to 5. Retrieving between 2 and 5 vectors showed minimal performance differences, with accuracy ranging from 89% to 91%. However, when retrieving only 1 vector, accuracy dropped to 85%, and the proportion of `None` outputs increased to 8%. We acknowledge that this behavior may be influenced by our controlled workload and limited test distribution. Nonetheless, as the results indicate, even if the encoding mechanism is imperfect, retrieving multiple similar vectors enables the LLM to generate more robust and contextually grounded explanations, mitigating potential inaccuracies.

End-to-end Response Time. The end-to-end response time consists of three components: the encoding overhead from the smart router, the search time within the knowledge base, and the processing and generation overhead of the LLM. As mentioned in Section 3.2, our smart router is lightweight, with average inference time lower than 0.1ms. Since our knowledge base is currently small (with only 20 queries), the search time per request also remains under 0.1ms. If the knowledge base grows in size, the search time will inevitably increase, but we do not expect this component to dominate, given recent advances in vector indexing [10]. The LLM's processing (thinking) time is generally fast ($\leq$ 2 seconds), but average generation time is around 10 seconds. This timing balance highlights that while retrieval is near-instantaneous, the generation step requires more time. While there is room for further improvement, the end-to-end response time is acceptable because the output is meant to be consumed by users.

Our experiments were conducted using both Doubao and ChatGPT 4.0, and we observed minimal differences in accuracy and the end-to-end response time between them. A comprehensive analysis of different language models is a future direction.

An additional advantage of using an LLM is its flexibility in offering a conversational interface that allows follow-up questions. For instance, in this example, a user might inquire why the predicate on the `customer` table does not benefit from the index on `c_phone`. The LLM can provide an in-depth explanation, clarifying that many database systems cannot utilize indexes on columns when functions like `substring` are applied directly to the indexed column.

6.3 Participant Study

To evaluate users' perceptions of our explanation quality, we designed a human-subject study focused on measuring ease of understanding. To ensure a fair comparison, we divided participants equally into two groups. Both groups were given the same query, as shown in Example 1, along with essential contextual information (e.g., the purpose of the survey and an overview of the hybrid engine structure).

The first group received both the AP and TP plan details (presented in JSON format for better readability) along with the LLM-generated explanation. We asked users to review both the

[4] During evaluation, three experts independently voted "agree" or "disagree" on each generated explanation. In 98.5% of the queries, all three experts reached unanimous decisions. For the remaining 1.5%, consensus was achieved after discussion. This high level of consistency reinforces the reliability of the expert evaluations used to validate the accuracy of the LLM-generated explanations.

Table 4: Comparison between our method and DBG-PT

Metric	Our Method	DBG-PT
Accurate Explanations	91%	67%
None Rate	3.5%	2%
Wrong Explanations	5.5%	31%

plan details and the LLM explanation, record the time taken until they indicated full understanding of the explanation. Then we ask them to submit their interpretations. The second group initially received only the AP and TP plan details, and we similarly recorded the time they spent to fully understand the performance differences based solely on these plan details. Users were also asked to submit a brief description of their understanding to assess correctness. Then, we further provided them with the LLM-generated explanation and asked if they would like to modify or adjust their initial understanding based on this new information. Finally, we asked them to rate the difficulty of understanding both the original plan details and the LLM explanation on a scale from 0 (easiest) to 10 (hardest). Since both groups spent time reviewing the plan details, the difference between two measured time reflects only the additional cognitive effort required to analyze and interpret the plans with or without the LLM explanation. This design fairly isolates the impact of structured versus natural language output on user understanding.

Our results are as follows. For participants who did not initially receive the LLM-generated explanation, 60% correctly identified the reason for the plan performance differences between plans, with an average time of 8.2 minutes (including the time spent on reading and understanding the plan details). The remaining 40% submitted incorrect reasons; however, after reviewing the LLM's generation, they were able to correct their understanding. The average difficulty rating for understanding the plan details was 8.5, while the LLM-generated explanation received an average difficulty rating of 3. For participants who received the LLM-generated results from the start, the average time taken to understand the reason was 3.5 minutes, and all users in this group were able to summarize the correct reason. These results indicate that the explanation by LLM reduces both the time required for understanding and the perceived difficulty, enhancing the comprehension of query performance differences.

6.4 Comparison with Other Methods

We additionally compare our approach with DBG-PT [8], which aims to suggest hints for debugging regressions in query execution time. DBG-PT leverages LLMs to identify and reason about structural differences between query plans from the same engine. In a sense, DBG-PT can be viewed as a baseline similar to our method but without RAG. We only provide the same TP and AP plan details for DBG-PT, without any historical queries or expert explanations. Specifically, we adjust the prompts in Table 1 by removing RAG-related context in the task description while retaining the same background information, question, and any additional user-provided prompts to the LLM. We use the same 200 synthetic queries to test DBG-PT, and the generation is manually evaluated by the same three database experts. Comparison results are shown in table 4.

Even without RAG, DBG-PT still demonstrates strong capability in analyzing structured plans by carefully comparing their differences. However, it shows less accuracy in its explanations for plan performance differences. As an illustrative example, Table 3 presents the generated results for the query shown in Example 1. In this case, the DBG-PT explanation exhibits some inaccuracies, such as misinterpreting index usage and overemphasizing minor factors while missing critical execution details. Building on this example and broader observations across additional test queries, we summarize the key limitations of DBG-PT as follows:

- *Fundamental errors:* It may misinterpret index usage. For instance, when a query includes a predicate like `substring (c_phone,1,2) in (...)`, no index is used; however, it still assumes AP is faster due to perceived index benefits.
- *Overemphasis on minor factors:* DBG-PT often overemphasizes column-oriented storage as the key reason for AP's speed, while underemphasizing critical factors such as TP's lack of indexes or inefficient join methods.
- *Ignoring limitations:* Despite instructions to avoid comparing costs between AP and TP, DBG-PT still seems to rely on cost differences sometimes, which is problematic because these costs are calculated differently and do not correlate well with real execution latencies.
- *Lack of context for relative values:* Without the RAG-enriched input, DBG-PT struggles to assess the significance of certain values without experience. For example, it cannot determine whether the size of an `OFFSET` or `LIMIT` is large enough to impact plan efficiency without historical execution data.

7 Conclusion and future work

In conclusion, our study demonstrates the effectiveness of a RAG-augmented LLM framework in providing user-friendly, accurate explanations for query performance in HTAP systems. By leveraging expert knowledge through a knowledge base containing past explanations, our approach enhances the accuracy and relevance of LLM-generated explanations, allowing users to better understand complex execution plans without needing specialized expertise. This framework effectively balances efficiency and accuracy by utilizing pre-trained public models with targeted context augmentation, reducing reliance on costly expert intervention.

Several areas remain open for future work, including developing strategies for maintaining the knowledge base (e.g., selecting representative queries and expiring stale queries), establishing benchmarks and automated tools to evaluate explanation quality. The promising results from our work demonstrate the potential of RAG-augmented LLM in enabling more reliable, scalable, and intelligent solutions for automated database performance analysis.

8 Artifacts

This research was conducted internally at ByteDance, Inc. (San Jose, CA). While we apologize that we are unable to release the full materials of the HTAP system, we have open-sourced an example implementation and a subset of the data at https://github.com/Hap-Hugh/LLM-Explain-HTAP for the approach described in this paper.

Acknowledgments

We thank the members of the ByteDance US Infrastructure System Lab for their valuable feedback, discussions, and support throughout the development of this work. We are especially grateful to Kui Wei, Shangyu Luo, Peizhi Wu, and Yukun Huang for their insightful suggestions, which significantly improved the system's design and presentation.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073* (2022).

[3] ByteDance. 2024. Doubao Large Language Model, https://www.doubao.com/. https://www.doubao.com/

[4] Jianjun Chen, Yonghua Ding, Ye Liu, Fangshi Li, Li Zhang, Mingyi Zhang, Kui Wei, Lixun Cao, Dan Zou, Yang Liu, et al. 2022. ByteHTAP: bytedance's HTAP system with high data freshness and strong data consistency. *Proceedings of the VLDB Endowment* 15, 12 (2022).

[5] Jianjun Chen, Li Zhang, Yu Xie, Wei Ding, Lixun Cao, Ye Liu, Yonghua Ding, Fangshi Li, Ke Wu, Haibo Xiu, et al. 2025. Vedb-htap: a Highly Integrated, Efficient and Adaptive HTAP System. *Proceedings of the VLDB Endowment* 18, 12 (2025).

[6] Jonathan Fürst, Catherine Kosten, Farhad Nooralahzadeh, Yi Zhang, and Kurt Stockinger. 2025. Evaluating the Data Model Robustness of Text-to-SQL Systems Based on Real User Queries. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, 158–170. doi:10.48786/EDBT.2025.13

[7] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[8] Victor Giannakouris and Immanuel Trummer. 2024. DBG-PT: A Large Language Model Assisted Query Performance Regression Debugger. *Proceedings of the VLDB Endowment* 17, 12 (2024).

[9] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[10] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

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

[12] Anna Mitsopoulou and Georgia Koutrika. 2025. Analysis of Text-to-SQL Benchmarks: Limitations, Challenges and Opportunities. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, 199–212. doi:10.48786/EDBT.2025.16

[13] Ananya Rahaman, Anny Zheng, Mostafa Milani, Fei Chiang, and Rachel Pottinger. 2025. Evaluating SQL Understanding in Large Language Models. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, 909–921. doi:10.48786/EDBT.2025.74

[14] Vikramank Singh, Kapil Eknath Vaidya, Vinayshekhar Bannihatti Kumar, Sopan Khosla, Murali Narayanaswamy, Rashmi Gangadharaiah, and Tim Kraska. 2024. Panda: Performance debugging for databases using LLM agents. (2024).

[15] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[16] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025. DBCopilot: Natural Language Querying over Massive Databases via Schema Routing. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, 707–721. doi:10.48786/EDBT.2025.57

[17] Xianghong Xu, Zhibing Zhao, Tieying Zhang, Rong Kang, Luming Sun, and Jianjun Chen. 2023. COOOL: A Learning-To-Rank Approach for SQL Hint Recommendations. *5th International Workshop on Applied AI for Database Systems and Applications* (2023).

[18] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2023. D-bot: Database diagnosis system using large language models. *arXiv preprint arXiv:2312.01454* (2023).

[19] Xuanhe Zhou, Xinyang Zhao, and Guoliang Li. 2024. LLM for Data Management. *Proceedings of the VLDB Endowment* 17, 12 (2024).

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