Evaluating the Practical Effectiveness of LLM-Driven Index Tuning on Microsoft SQL Server

Xiaoying Wang
wangxiaoying@microsoft.com
Microsoft Research
Redmond, USA

Wentao Wu
wentao.wu@microsoft.com
Microsoft Research
Redmond, USA

Vivek Narasayya
viveknar@microsoft.com
Microsoft Research
Redmond, USA

Surajit Chaudhuri
surajitc@microsoft.com
Microsoft Research
Redmond, USA

ABSTRACT

Indexes are crucial for database performance. Index tuning, i.e., selecting appropriate indexes for a database workload, is an important problem. The state-of-the-art index tuning tools in the industry, e.g., Database Tuning Advisor (DTA) developed for Microsoft SQL Server, rely on a "what-if" API, which can estimate the cost of a query for a given index configuration. They take as input a SQL workload and constraints such as a storage bound, and search over the large space of index configurations to find one with low optimizer-estimated cost for the input workload. Large language models (LLMs) offer a different approach to index tuning, using knowledge they have learned from publicly available training data. However, the effectiveness of LLM-driven index tuning in comparison to today's index advisors, particularly on enterprise workloads, remains unclear.

In this paper, we study the practical effectiveness of LLM-driven index tuning on Microsoft SQL Server using both industrial benchmarks and real-world enterprise customer workloads, and compare it with DTA. Our results show that while LLMs in several cases identify configurations that significantly outperform those found by DTA in terms of execution time, they suffer from high variance in index recommendation quality. Furthermore, index recommendations from the LLM are often substantially worse than DTA in terms of optimizer-estimated cost, making it challenging to extend cost-based index advisors such as DTA to leverage LLMs for index tuning. We point to some areas of future work that may be important for robustly leveraging LLMs for index tuning.

PVLDB Reference Format:
Xiaoying Wang, Wentao Wu, Vivek Narasayya, and Surajit Chaudhuri. Evaluating the Practical Effectiveness of LLM-Driven Index Tuning on Microsoft SQL Server. PVLDB, 19(12): 4049 - 4062, 2026.
doi:10.14778/3827998.3828015

1 INTRODUCTION

Index tuning, the problem of deciding which indexes to create for a given database workload, is an important and challenging problem that has been studied extensively [8, 12, 13, 17, 35, 54, 59, 60, 64, 68, 69, 71, 77]. Today's index tuning tools, aka *index advisors* deployed by commercial and open-source database systems such as Microsoft SQL Server [2, 13], IBM DB2 [64], and PostgreSQL [33], adopt a cost-based architecture where the goal is to minimize the cost of the input SQL workload *estimated* by the database query optimizer using a "what-if" API. Although these cost-based index advisors have been proven effective in practice, inaccuracies in cost estimates can lead to suboptimal index recommendations [20, 74], sometimes even resulting in query performance regressions (QPRs) [16, 75]. To mitigate the impact of inaccurate cost estimates on index tuning, previous work has explored techniques such as QPR detection or improvements to underlying cost models [20, 45, 48, 75]. However, the effectiveness of these remedies remains limited due to fundamental challenges in cost estimation [31, 37, 44].

Large language models (LLMs) offer an alternative approach to index tuning. LLMs have shown promise in other database performance tuning tasks, such as query rewriting [19, 41, 43, 46, 50, 82] and query optimization [6, 62, 81], suggesting that it can capture and apply useful domain knowledge without explicit cost models. Several recent studies have taken initial steps toward the application of LLMs to index tuning [25, 40, 84], exploring different task formulations and prompt designs, and reporting promising early results. However, these studies primarily evaluated LLMs on open benchmarks that are likely included in LLM's training data. As a result, the effectiveness of LLMs when applied to real-world enterprise workloads is not well understood, particularly those involving complex queries with views, common table expressions (CTEs), nested sub-queries, etc. or databases with a large number of manually created indexes — characteristics that open benchmarks typically lack. Moreover, the benefits of LLMs compared to a state-of-the-art commercial index tuner, such as Microsoft SQL Server Database Tuning Advisor (DTA) [12], remain unclear, as the evaluations in previous work were limited to PostgreSQL and comparison with simplified baselines.

In this paper, we study LLM-driven index tuning using both public benchmarks and *real-world enterprise customer workloads*. We evaluate its effectiveness in comparison with DTA on Microsoft SQL Server. We focus on pre-trained LLMs in an end-to-end setting, given only basic information about the database and workload. Unlike previous work that relies mainly on estimated cost from the query optimizer [35, 85], our evaluation is based on the actual *execution time* of each query. Our study covers the following aspects:

- *Single-query vs. Multi-query Workloads.* We evaluate index tuning for both single-query and multi-query workloads, both of which are important scenarios in practice.
- *Constrained vs. Non-constrained Tuning.* Constraints (e.g., the maximum number of indexes or storage space allowed) play an essential role in index tuning as they can change the search space of the index tuner. Both constrained and non-constrained index tuning have applications in practice [13, 35, 64, 77].

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828015

- *Synthetic Benchmarks vs. Real-world Workloads.* Existing evaluations are largely based on public benchmarks, which are available for LLM training. Real-world enterprise customer workloads, on which LLMs have not been trained, therefore provide a new test to assess the effectiveness of LLM-driven index tuning.

In our experiments, we invoke the LLM five times for each query or workload. For each invocation, we implement the LLM's recommendation and execute the query/workload. Due to non-determinism inherent in LLMs, we observe that even for the same query, LLM's recommendations change significantly across invocations. Note that we invoke DTA only once since it is deterministic. The key findings from our experimental study are:

LLM-recommended indexes provide complementary benefits compared to DTA. When considering the *best* recommendation across the five invocations, for a significant fraction of single-query workloads (approximately 67%), we find that the LLM can identify a configuration that is comparable to that returned by DTA (Section 3.1). These results hold for both industry benchmarks and real-world workloads. We also find that the LLM often proposes fewer indexes than DTA (Section 3.2). Moreover, for 31% of queries, the LLM identifies configurations that substantially outperform those recommended by DTA ($\geq$ 20% faster). Such queries are primarily those where DTA's recommendation is misled by inaccurate optimizer cost estimates (Section 3.3). For multi-query workloads, DTA delivers more consistent performance improvements overall compared to the LLM. However, the LLM still suggests configurations that significantly outperform DTA in two of the five workloads evaluated in our experiments (Section 5.1).

LLM exhibits high performance variance, with worst-case outcomes significantly worse than DTA. LLM-driven index tuning exhibits substantial performance variance across invocations for the same query, and its worst-case outcomes can significantly underperform DTA. In some cases, these recommendations can also lead to severe QPRs (Sections 3.4 and 5.1). This variance occurs across workloads as well. For example, in some workloads, even the *best* index configuration found by the LLM is significantly worse than DTA. We also observed that poor LLM recommendations became more frequent as workload size increases (Section 5.3). These results suggest that, despite their promise, it is not prudent to directly adopt LLM-generated recommendations in practice. The next two findings are motivated by the goal of evaluating if LLMs can be *robustly* used in index tuning, i.e., without incurring the high risk of performance degradation.

Directly integrating LLM recommendations into DTA often leads to performance degradation. We evaluate the feasibility of expanding DTA's pool of candidate indexes with LLM-recommended indexes, while leaving the rest of DTA unchanged. In principle, this approach expands DTA's search space and may allow it to identify configurations with lower estimated costs while retaining the deterministic behavior of DTA. However, for single-query workloads, our results show that this strategy is ineffective in practice. The final configurations selected by DTA from the augmented pool of candidates is often unchanged, or leads to performance degradation when implemented, again due to inaccuracies in the optimizer's cost estimation (Section 6.1). For multi-query workloads, the LLM tends to directly identifies indexes that benefit multiple queries. This is in contrast to DTA, which first optimizes each query independently to identify candidates and then merges candidates from different queries to include new candidates that are promising at the workload level. Indexes identified by the LLM are not always considered by DTA and, in two out of the five workloads, expanding DTA's pool of candidate indexes with LLM-recommended indexes improves DTA's quality significantly (Section 5.2 and 6.1).

Table 1: Summary of database and workload properties.

Name	DB Size	# Queries	# Tables	# Joins	# Scans	# Ops (Avg/Max)
TPC-H	sf=10	22	8	2.8	3.7	19.1 / 38
Real-D	587GB	29	7912	15.6	17	71.7 / 197
Real-M	26GB	40	474	20.2	21.7	71.7 / 183
Real-R	100GB	24	20	6.5	7.2	39.3 / 210
Real-S	40GB	12	32	7.3	9.7	40.9 / 67

Effective heuristics can be distilled from LLM. When analyzing LLMs' recommendations and their reasoning process on single-query workloads, we observed a few recurring patterns in their choice of indexes (Section 4.1). Based on this observation, we develop a simple rule-based index tuner that aims to distill these patterns into a deterministic program (Section 4.2). Interestingly, we find that the heuristic index tuner provides most of the benefits of LLM-recommended indexes, particularly for queries where DTA underperforms LLM (Section 4.3). Although more experiments are required to evaluate whether this approach is effective across different prompts and models, this experiment serves as a "proof-of-concept" that meaningful value can be derived from analyzing LLM's reasoning without necessarily incurring the high variance that accompanies LLMs.

In the following, we summarize the implications of our study for database practitioners and researchers and point to opportunities for future work.

- Although LLMs demonstrate potential for providing good-quality index recommendations on industry benchmarks and real-world enterprise workloads, due to the high variance in quality, *safely* realizing their potential in practice remains an open problem.
- A straightforward integration of LLM-recommended indexes into cost-based index tuning advisors such as DTA is ineffective in practice. Exploring alternative ways of leveraging the best of both approaches, e.g., with prompts that may help limit the downside, or using index advisors as a tool and having LLM play the role of an orchestrator, is an interesting area of future work.
- *Validation* of index recommendations has previously been used to prevent regressions by evaluating a configuration after deployment and workload execution [10, 75]. However, index creation and workload execution can be expensive and disruptive. Since LLMs can identify high-quality indexes but exhibit high variance, validation is essential for leveraging LLM-driven index tuning. Thus, developing more efficient and less disruptive validation mechanisms, such as those enabled by branching techniques [53], is an important direction for future index tuning.
- Another direction is to harness the potential of LLM-driven index tuning while reducing its variance. One approach is to develop heuristic techniques by incorporating domain knowledge distilled from LLMs. It is also interesting to investigate whether models fine-tuned for index tuning can outperform pre-trained LLMs by achieving more consistent quality.

```
You are a database expert for Microsoft SQL Server with index tuning capabilities. Your goal is
to recommend non-clustered indexes only on Microsoft SQL Server. Find below:
1. A SQL query whose performance you are trying to improve.
2. Schema information for the tables and views used in the query, including existing indexes on
the tables and definition of the views referenced in the query.
3. The plan for the query in tabular format where each row corresponds to a physical operator in
the plan. Each row contains information about the operator such as estimated rows estimated cost.
Your task is to analyze the query, schema and execution plan and recommend zero or more non-
clustered indexes that should be created on the tables referenced in the query to significantly
improve this query's performance.

1. SQL Query:
{{ QUERY }}

2. Schema information:
{{ SCHEMA INFO }}

3. Execution plan:
{{ PLAN INFO }}

Your response should strictly consist of a python-compatible JSON response of the following
schema. Do not include any additional text in the prompt, such as descriptions or intros. Return
just a python-compatible list of SQL commands only.
{
      commands: [list of the SQL commands]
}
If the query references a view, do not recommend indexes on the view, but rather on the
underlying tables of the view. The underlying tables of the view can be found in the schema
information. If a table is only scanned once, do not recommend multiple indexes on it. All the
referenced columns for each table can be found in the schema information. Do not recommend
filtered indexes.


Think hard and step by step!
```

Figure 1: Single-query workload prompt template.

2 METHODOLOGY

In this section, we present the methodology used in our study.

2.1 Benchmark and Customer Workloads

Unlike existing work that has been primarily evaluated using public benchmarks with synthetic data and queries, we focus on real-world customer workloads in our study. Table 1 summarizes the properties of the workloads that we used, including schema complexity (e.g., the number of tables), query complexity (e.g., the average and/or maximum number of joins, table scans, and operators per query), and the sizes of database and workload.

We adopt four customer workloads with diverse properties. These workloads consist of real-world analytical queries that make intensive use of features such as CTEs and views, and the underlying tables contain numerous pre-existing indexes that were manually created by human experts. We also include the standard **TPC-H** benchmark with a scaling factor (sf) of 10.

2.2 Experimental Setups

2.2.1 Evaluation Baseline. To evaluate the effectiveness of LLM-driven index tuning on real-world queries, we adopt Microsoft Database Tuning Advisor (DTA) [2, 12] as the baseline in this study. DTA is a commercial industrial-strength index tuner that represents the current state of the art (SOTA) in classic cost-based index tuning. Thus, it serves as a strong reference point for comparing LLM-driven index tuning.

DTA adopts a cost-based software architecture that contains three major components: (1) *workload parsing and analysis*, which analyzes input workload queries to identify *indexable columns* [13], such as columns appearing in filter or join conditions; (2) *candidate index generation*, which determines the key columns and included columns of potentially beneficial indexes as well as the orders of the columns; and (3) *configuration enumeration*, which selects a subset (a.k.a. a configuration) from the candidate indexes that can minimize the estimated workload execution cost while adhere to the given index tuning constraints. To estimate the cost of a query (and thus the workload) with an index configuration, DTA uses the "what-if" API [14], which is an extended functionality of the query optimizer

Node Id	Parent Node Id	Operator	Arguments	Estimated Cost	Estimated Rows
1		Aggregate	**GroupBy**: o_orderpriority; **AggExpr**: COUNT(*)	925.999	5
2	1	Sort	**OrderBy**: o_orderpriority	925.999	80
3	2	Hash Match		925.959	80
4	3	Nested Loops		925.298	456358
5	4	Index Scan	**Table**: orders; **Where**: o_orderdate >= '1997-03-01' AND o_orderdate < '1997-06-01'	169.076	456358
6	4	Index Seek	**Table**: lineitem; **Where**: l_commitdate < l_receiptdate	754.768	1

Figure 2: Prompt plan example (TPC-H q04, simplified).

that can perform this cost estimation *without* materializing the indexes to the underlying storage system.

Recent studies have proposed various extensions to DTA such as reducing query performance regression [20, 72] and improving its scalability when processing large workloads [58, 60, 68, 69, 77]. These techniques have not been used in the production DTA studied in this paper, and their integration with DTA remains future work.

2.2.2 Performance Metric. Rather than using estimated cost [35, 85], we adopt execution time as our main evaluation metric. For each configuration, we execute each query in isolation five times, with each run capped at five minutes (300 seconds), and report the median execution time. For multi-query workloads, we use the total execution time of all queries in the workload.

2.2.3 Hardware and Software Platform. We use Microsoft SQL Server 2022 as the database system for evaluation. All experiments are conducted on a standard Azure M32bds v3 virtual machine [49] with 32 vCPUs and 256 GB memory, running Windows Server 2022.

2.3 Setups of LLM-Driven Index Tuning

We tried various LLMs, such as DeepSeek-R1 [29], Qwen3 [79], GPT-4o [30], and GPT-5 [51]. We observe substantial variance in the effectiveness of different LLMs, with GPT-5 performing the best among all models evaluated in our experiments. Therefore, *all evaluation results reported in this paper are based on GPT-5*. For each prompt, we invoke GPT-5 five times independently and collect the responses. In the remainder of this paper, *we use the two terms GPT-5 and LLM interchangeably*.

2.3.1 Single-query Workload Tuning. Figure 1 presents the prompt template for tuning single-query workloads. It begins with instructions that describe the input information provided and the optimization task. It ends with specifications of the output format and additional constraints, such as prohibiting index creation on views. Query-specific information that needs to be included in the prompt consists of the following: (1) the SQL text of the query, (2) the database schema, and (3) the current query execution plan based on the original index configuration. This information is represented by the placeholders $\{\{\cdots\}\}$ in Figure 1.

Database Schema. The information about the database schema contains all *base tables* that are referenced by the query, along with their cardinalities, included columns, and *pre-existing indexes*. Definitions of all *views* referenced by the query are also included.

Query Execution Plan. Following previous work [50], the query plan with the original index configuration (i.e., pre-existing indexes) is represented in tabular form, where each row corresponds to a *physical* operator and includes its detailed description (e.g., filter or join condition), estimated cost, and estimated cardinality. To ensure a fair comparison with DTA, query plans are obtained using

```
You are a database expert for Microsoft SQL Server with index tuning capabilities. Your
goal is to recommend non-clustered indexes only on Microsoft SQL Server. You are given a
workload consisting of {{ WORKLOAD SIZE }} queries. For each query, find below:
1. A SQL query whose performance you are trying to improve.
2. Schema information for the tables and views used in the query, including existing
indexes on the tables and definition of the views referenced in the query.
3. The plan for the query in tabular format where each row corresponds to a physical
operator in the plan. Each row contains information about the operator such as estimated
rows estimated cost.
Your task is to analyze the workload, including the sql query, schema and execution plan
of each query and recommend zero or more non-clustered indexes that should be created on
the tables referenced in the workload to significantly improve this workload's
performance.

###### Query 1 ######
1. SQL Query:
{{ QUERY }}

2. Schema information:
{{ SCHEMA INFO }}

3. Execution plan:
{{ PLAN INFO }}

###### Query 2 ######
1. SQL Query:
{{ QUERY }}

2. Schema information:
{{ SCHEMA INFO }}

3. Execution plan:
{{ PLAN INFO }}

……

Your response should strictly consist of a python-compatible JSON response of the
following schema. Do not include any additional text in the prompt, such as descriptions
or intros. Return just a python-compatible list of SQL commands only.
{
      commands: [list of the SQL commands]
}
If a query references a view, do not recommend indexes on the view, but rather on the
underlying tables of the view. The underlying tables of the view can be found in the
schema information. Do not recommend filtered indexes.
Optimize the performance of the entire workload, but use at most {{ BUDGET }} indexes.

Think hard and step by step!
```

Figure 3: Multi-query workload prompt template.

the Microsoft SQL Server command "SET SHOWPLAN_XML ON" without executions. As an example, Figure 2 shows the query plan representation for **TPC-H** query 4. The plan uses a nested loop followed by a hash match to implement the semi-join, then sorts and aggregates the qualifying rows. Details such as parallelism are omitted in this example for the sake of clarity.

Remark. Our goal is not to identify an optimal prompt, but to provide LLM with basic information that a human expert would need to make informed index recommendations. In this study, we focus on evaluating the fundamental capability of LLM for index tuning, given only rudimentary information. More variations of LLM-driven index tuning are discussed in Section 7.

2.3.2 Multi-query Workload Tuning. For each query in the workload, we generate query-specific information using the same procedure as for single-query workloads and then concatenate these representations. Figure 3 presents the prompt template used for tuning multi-query workloads. We make a few modifications to the instructions to accommodate the multi-query tuning scenario, incorporating additional information such as the workload size and the number of indexes allowed.

Unlike existing work [25, 84] that compresses input workload information to stay within LLM's token limit, we deliberately retain the complete information for each query in the prompt, for two reasons. First, GPT-5 supports up to one million tokens, which is sufficient for many real-world customer workloads. Second, input compression may eliminate critical details of the workload, resulting in degraded performance and obscuring the true capabilities of LLM that we aim to evaluate.

3 SINGLE-QUERY WORKLOADS

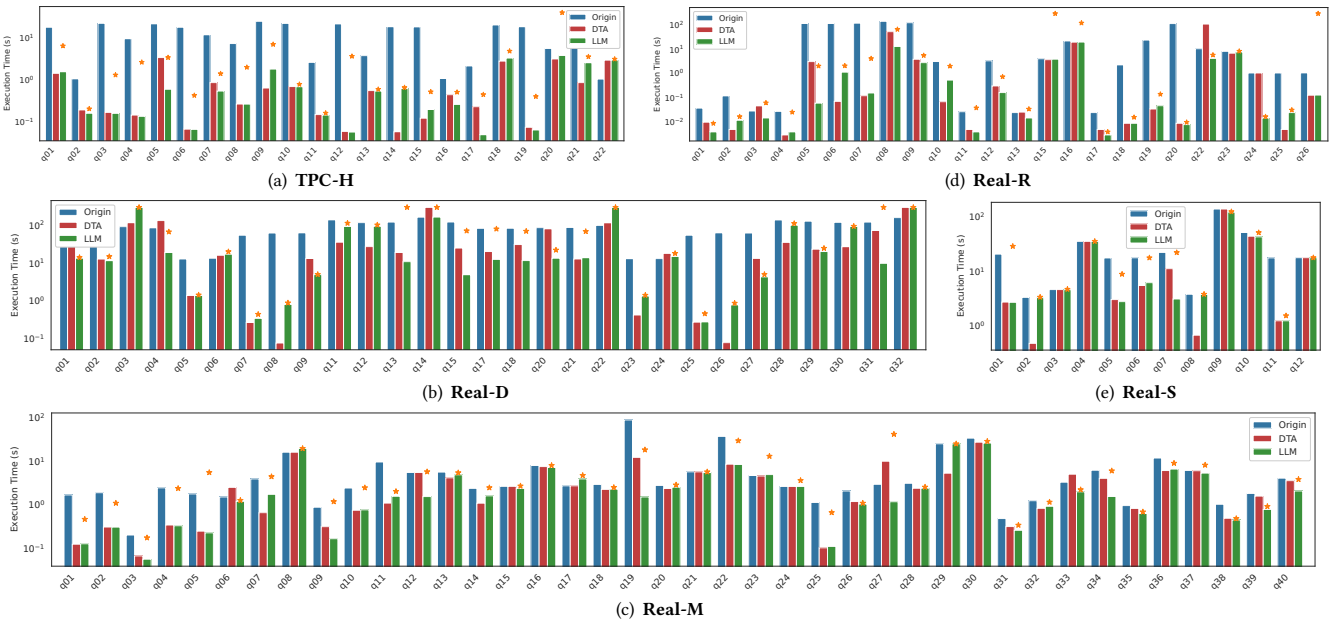
In real-world applications, single-query workloads are common. For example, Microsoft SQL Server has a feature such that the query optimizer can automatically suggest "missing indexes" for a given query that would significantly improve the query performance if these indexes were created [1]. Moreover, they are special cases of more general multi-query workloads that will be studied in Section 5. In fact, DTA uses a *two-phase greedy search* algorithm [13, 77] where it breaks the overall index selection problem into two search phases: (1) query-level search and (2) workload-level search. The query-level search phase addresses exactly the single-query workload tuning problem.

We do not enforce constraints such as the maximum storage space allowed for LLM on single-query workloads. By default, DTA sets the storage space constraint as 3× of the database size, which is more than sufficient for any single query and would not result in premature recommendations due to space constraints.

3.1 Overview of Evaluation Results

Figure 4 presents the evaluation results when comparing GPT-5 with DTA for tuning single-query workloads, in terms of query execution time with the recommended indexes. To illustrate the best potential of LLM, in the analysis from Section 3.1 to Section 3.3, we use the best-performing recommendation among the five LLM responses (denoted by the green bar in Figure 4) as the outcome of LLM-driven index tuning. We also discuss worst-case results in Section 3.4 but defer other results (e.g., from the first and median responses) to the full version of the paper [67]. The average response time of GPT-5 is 1.1, 1.7, 1.9, 2.2, and 2.0 minutes on queries in **TPC-H**, **Real-M**, **Real-D**, **Real-R**, and **Real-S**, respectively.

- **TPC-H**: Compared to DTA, GPT-5 performs similarly or better for most queries. For example, it significantly outperforms DTA for queries 5 and 17, whereas DTA is advantageous for queries 14 and 21. Since **TPC-H** is a well-known benchmark that is publicly available, it may have been seen during GPT-5's training. Therefore, we also tried to "anonymize" the queries by replacing the table and column names with randomly generated identifiers. However, we observed similar performance for GPT-5 on the anonymized version, which is consistent with the findings reported in prior work [25].
- **Real-D**: We observe that LLM significantly outperforms DTA for several queries (e.g., queries 1, 4, 9, 15, 18, 20, 27, and 31), while DTA performs better for others (e.g., queries 3, 11, 22, 23, 26, 28, and 30). Overall, the observations are similar to those from **TPC-H**, despite the substantially higher schema and query complexity of **Real-D**.
- **Real-M**: DTA achieves better performance for only a small number of queries (e.g., queries 7 and 29), whereas LLM outperforms DTA for many others (e.g., queries 6, 9, 12, 19, 27, and 33). Moreover, for several queries (e.g., queries 6, 27, and 33), indexes recommended by DTA lead to query performance regressions (QPRs) [20, 21, 73, 75]. In these cases, LLM is able to identify a better plan that improves query execution time.
- **Real-R**: The observations are similar to those from **Real-M**. Except for a few queries (e.g., queries 6, 10, and 25), LLM outperforms DTA, sometimes quite significantly (e.g., for queries 5 and



(a) **TPC-H** (d) **Real-R**

(b) **Real-D** (e) **Real-S**

(c) **Real-M**

Figure 4: LLM-driven index tuning (best) vs. DTA for tuning single-query workloads (marker denotes the worst run of LLM).

22). Notably, DTA's recommendation results in a severe QPR for query 22 (nearly a 10× slowdown in execution time), whereas LLM improves by approximately 60%, reducing its execution time from 10 seconds to 4 seconds.

- **Real-S**: LLM and DTA perform similarly for **Real-S** queries. DTA significantly outperforms LLM for queries 2 and 8, while LLM significantly outperforms DTA for query 7. We do not observe QPRs for the **Real-S** queries.

Based on the observations, we have the following conclusion.

Key Finding 1. *With a small number of trials, the best outcome of LLM-driven index tuning underperforms, matches, or substantially outperforms DTA across different sets of queries, each comprising a considerable fraction of our test cases.*

This observation highlights the complementary potential of LLM-driven index tuning for industrial single-query workloads, as it can identify configurations that substantially outperform those produced by DTA for many queries, particularly in cases where DTA's recommendation leads to a performance regression.

3.2 Efficacy of LLM-Recommended Indexes

Since we impose no explicit constraints on the number of indexes recommended by LLM, its strong performance may stem from proposing a sufficiently large set of indexes, thus increasing the likelihood of covering optimal configurations. To assess this possibility, we conduct a drill-down analysis to better understand the real efficacy of LLM-recommended indexes, as reported in Section 3.1.

Figure 5 presents the number of LLM-recommended indexes utilized by query plans for single-query workloads. For comparison, we also show the number of indexes recommended by DTA.

- **TPC-H**: For most queries, all indexes recommended by LLM are utilized by their query plans, with only a couple exceptions (e.g., queries 20 and 21). Interestingly, DTA recommends more indexes for certain queries (e.g., queries 2, 13, 14, 18, 20, and 21), which are all utilized by the corresponding query plans.
- **Real-D**: Most of the indexes recommended by LLM are utilized by the corresponding query execution plans. For more than half of the queries, DTA recommends substantially more indexes compared to LLM. For example, DTA recommends 17 indexes for the query 4 whereas LLM recommends only 7 indexes.
- **Real-M**: Similarly, DTA typically recommends considerably more indexes than LLM. However, it does not recommend any indexes for several queries (e.g., queries 8, 12, 15, 17, 21, and 24). Meanwhile, LLM recommends fewer indexes, most of which are utilized by the corresponding query execution plans.
- **Real-R**: DTA and LLM recommend similar numbers of indexes for most queries, with a few exceptions (e.g., queries 8, 9, and 12) where DTA recommends significantly more indexes.
- **Real-S**: Indexes recommended by LLM are not utilized by the corresponding query plans for several queries (e.g., queries 2, 3, and 4). Meanwhile, DTA does not recommend any indexes for quite a few queries (e.g., queries 3, 4, 9, and 12).

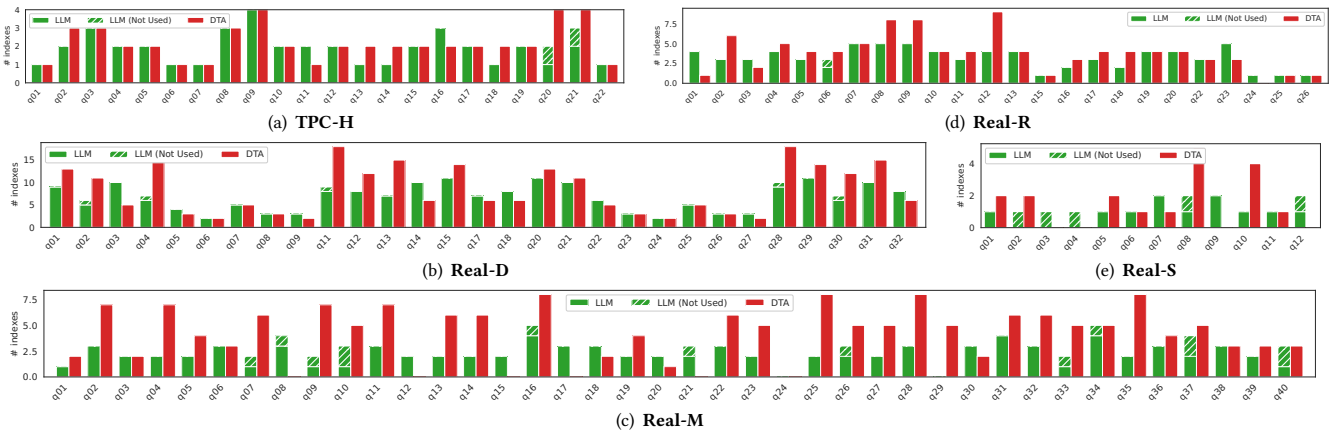
Based on the above observations, we conclude the following.

Key Finding 2. *Most of the indexes that LLM proposes for single queries are effectively utilized by the corresponding query plans. In many cases, LLM recommends even fewer indexes than DTA.*

3.3 What Happens When LLM Performs Better?

Next, we conduct a deeper investigation into the cases where LLM outperforms DTA. Specifically, we examine these queries by comparing the estimated costs of query plans produced using indexes recommended by DTA and LLM. Figure 6 summarizes the results. Only cases where LLM outperforms DTA are shown.

Each graph in Figure 6 is a histogram showing the distribution of queries by the difference in estimated costs between the plans produced by LLM and DTA. The x-axis denotes this cost difference: negative values indicate that the LLM-recommended indexes are favored by query optimizer's cost model, while positive values indicate the opposite. The red dashed line indicates zero difference.



(a) **TPC-H** (d) **Real-R**

(b) **Real-D** (e) **Real-S**

(c) **Real-M**

Figure 5: Comparison of index usage between LLM-driven index tuning and DTA for tuning single-query workloads.

The *y*-axis represents the number of queries that fall into each cost-difference bin. We have the following observation.

Key Finding 3. *For queries where LLM-recommended indexes yield faster execution time, their optimizer-estimated costs are consistently higher than those of the indexes recommended by DTA.*

This reveals the deployment risk shared by DTA and other cost-based index tuners. Although these cost-based index tuners are widely adopted in industrial applications [10, 16, 33, 64], their efficacy largely depends on the accuracy of the model used by the underlying query optimizer. It is well known that query optimizer's cost estimates can be inaccurate for various reasons, such as errors from cardinality estimation [31, 37, 66, 76]. Therefore, while these cost-based index tuners are generally reliable, they can sometimes result in poor index recommendations.

In contrast, the index tuning approach taken by LLM is based on reasonable rule-of-thumb heuristics (Section 4) derived from the world knowledge acquired during LLM's training. Such heuristics may not be comprehensive enough to cover all plan transformation rules built into a modern cost-based query optimizer [26], but they can be used to improve query execution. More importantly, although LLM can miss certain opportunities that DTA can find (e.g., when LLM underperforms DTA in Figure 4), its heuristic-based recommendation is *less susceptible to inaccurate cost estimates*. Table 2 presents such examples, where query plans using LLM-recommended indexes have a higher estimated cost but achieve a substantially better execution time.

Due to inaccurate cost estimates, DTA would not select the configurations identified by LLM since their estimated costs are larger (than the ones finally selected by DTA), even when they fall within the search space of DTA. Such inaccuracies can sometimes lead to severe QPRs, such as query 4 from **Real-D** and query 27 from **Real-M**. These results help explain the complementary performance of LLM-driven index tuning relative to DTA.

3.4 Performance Variance of LLM

So far, we have focused on the best results from LLM, highlighting its strong performance potential. However, we also observe substantial performance variance in LLM-recommended configurations. To further illustrate this, this section examines the *worst* outcomes among the five invocations, indicated by the star markers in Figure 4, and compares them against the best results.

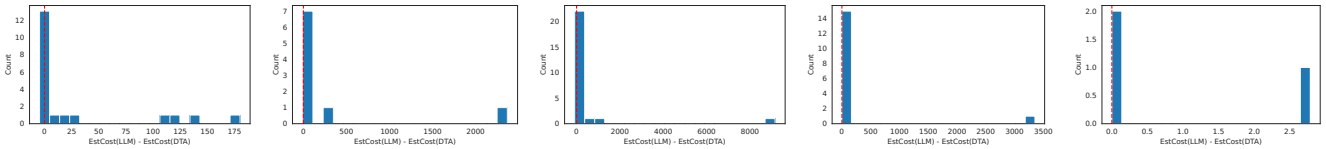
- **TPC-H**: The gap between the best and worst outcomes across the five LLM invocations is substantial for many queries (e.g., queries 1, 3, 4, 5, 8, 9, 12, and 20). This gap is particularly pronounced for query 20, where the worst outcome results in a severe QPR. With the worst-case outcomes, LLM no longer outperforms DTA for any query and, in most cases, performs substantially worse.
- **Real-D**: There are a few cases where the worst LLM response achieves a similar level of improvement to the best one and still outperforms DTA (e.g., queries 1, 9, 20, and 27). However, for the majority queries, the worst response offers no improvement in execution time. For some queries (e.g., queries 13, 14, and 31), it leads to QPRs and eventually timeouts, whereas the corresponding best response does not exhibit such behavior.
- **Real-M**: The execution time variance across different LLM invocations remains substantial. The worst responses lead to QPRs for queries such as 5 and 27. Moreover, they consistently underperform DTA, even for queries where the best LLM outcomes outperform DTA (e.g., queries 3, 9, 19, 27 and 34).
- **Real-R**: We observe two significant QPRs caused by the worst LLM responses, both resulting in timeouts. Although these worst-case outcomes can still accelerate some of the queries, they outperform DTA for only two queries (i.e., queries 22 and 24).
- **Real-S**: The worst of the five LLM responses cannot match the performance of DTA. In particular, for query 7, which is the only case where the best LLM response outperforms DTA, the worst response is unable to produce any improvement.

Key Finding 4. *LLM-driven index tuning exhibits substantial performance variance within a small number of trials. Its worst outcomes may significantly underperform DTA and, in many cases, lead to severe performance regressions.*

This observation motivates careful performance validation before adopting LLM-recommended indexes in production. We discuss this aspect in more detail in Section 6.2.

4 DISTILLING INSIGHTS FROM LLM

We found the effectiveness of LLM-recommended indexes on real-world single-query workloads surprising (Section 3). However, we also found that the non-determinism of LLM leads to significant variance (Section 3.4) in the index recommendation quality for the same query as well as across queries. We therefore conducted two follow-up studies. First, we analyzed the *reasoning process* output by



(a) **TPC-H** (b) **Real-D** (c) **Real-M** (d) **Real-R** (e) **Real-S**

Figure 6: Comparison of the estimated costs between LLM-driven index tuning and DTA for single-query workloads.

Table 2: Examples of inaccurate estimated cost with the what-if API vs. actual plan execution time.

Query	DTA Cost	DTA Time	GPT-5 Cost	GPT-5 Time
TPC-H q05	132.736	3.437s	268.35	0.603s
Real-D q04	1402.38	134.324s	3767.51	19.211s
Real-D q27	268.087	13.427s	295.872	4.375s
Real-M q19	2934.32	12.522s	12088.7	1.549s
Real-M q27	553.253	10.210s	1643.99	1.202s
Real-R q22	34.3663	111.259s	40.7258	4.268s
Real-S q07	5.64505	11.398s	8.43237	3.168s

the LLM during inferencing to characterize the basis on which LLM generates its recommendations. Our analysis reveals several rules of thumb used by the LLM when recommending indexes (Section 4.1). Second, using these rules of thumb, we built a simple *deterministic* index tuner. Our intent was to evaluate whether such an approach could derive most of the benefits of an LLM without inherting its variance in quality (Section 4.2).

4.1 Analysis of LLM's Reasoning Process

To analyze GPT-5's reasoning process, we instruct it to output its reasoning together with the recommended indexes. Figure 7 presents two typical examples. The recommended configurations result in a 97% improvement for **TPC-H** query 5 and a 95% improvement for query 7. We observe that LLM's reasoning usually begins with identifying the *bottleneck* operators in the input query plan, followed by analysis of how to use indexes to improve these bottlenecks. For example, for query 5, GPT-5 argues that having an index built on the orders table to seek by o_orderdate can replace the costly Clustered Index Scan operator and lead to significant speedup. Examining these LLM reasoning processes during our evaluation, we make the following observation.

Key Finding 5. *In summary, GPT-5's reasoning reveals several simple rules of thumb: (1) prioritize building indexes that reduce costly scan operations; (2) order index key columns according to cues from the query plan, such as filters, joins, and aggregates; (3) introduce covering indexes when possible to enable index-only scans and avoid expensive full table scans; and (4) ignore small table scans.*

These rules of thumb are much simpler than DTA's approach to index recommendation. DTA takes a cost-based approach that relies on the what-if API provided by the query optimizer. Even during candidate index generation, it makes cost-based decisions [12]. For example, after parsing the workload queries, DTA uses a *frequent pattern mining* procedure [3] to identify *interesting* table subsets and column groups based on their frequency and costs. Multi-column indexes are restricted to those defined on interesting column groups. For indexes on columns with filter conditions, DTA orders the columns by the estimated selectivity of the corresponding predicates. Moreover, DTA uses *index merging* [15] to combine similar candidate indexes, which is again guided by cost estimates.

```
Biggest costs come from Clustered Index Scan on orders (date filter) and full scan of
lineitem with bitmap. Add an index on orders to seek by o_orderdate and supply join
columns, and a covering index on lineitem keyed by l_orderkey to avoid scanning the
clustered table or doing massive lookups while providing l_suppkey, l_extendedprice, and
l_discount for the join and aggregation. Other tables are tiny or already well-indexed.
```

TPC-H q05 (improvement: 97%)

```
Most cost is from repeatedly probing lineitem by orderkey and then hitting the clustered
index to evaluate l_shipdate and fetch needed columns. Add a covering NCI on l_shipdate
to enable a selective range seek that can drive joins and reduce rows early. Also add a
covering NCI on l_orderkey so, if the optimizer keeps the current join order, it can
avoid clustered lookups and evaluate the shipdate predicate at the index level. Other
tables already have adequate supporting indexes; nation scans are negligible.
```

TPC-H q07 (improvement: 95%)

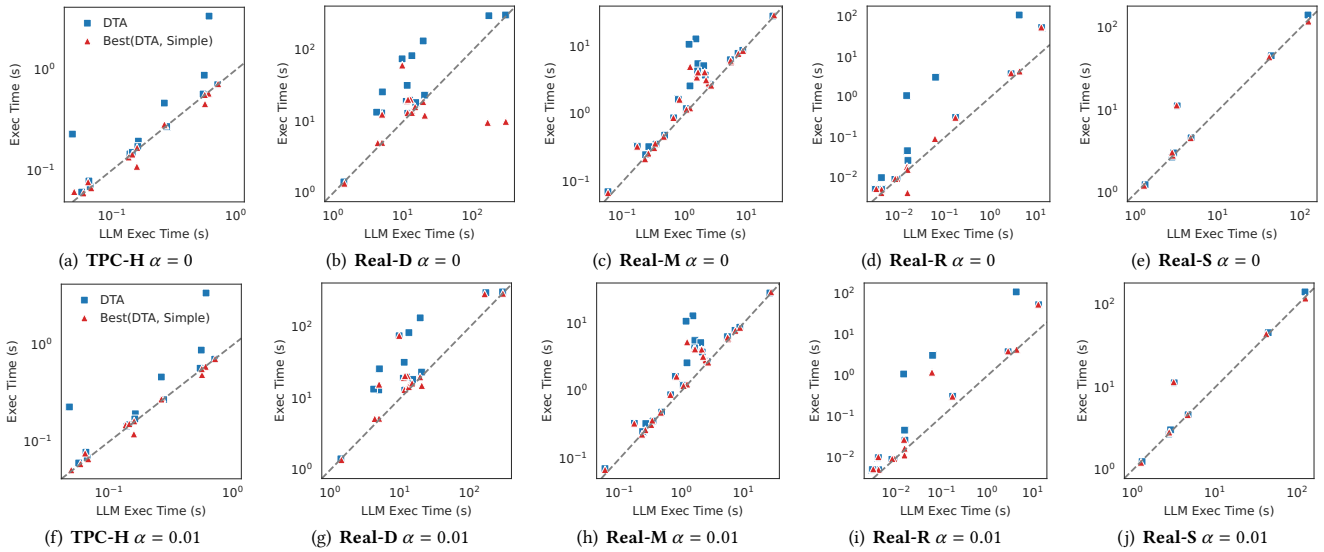
Figure 7: Examples of GPT-5's reasoning process.

4.2 A Simple Rule-based Index Tuner

These rules of thumb that we observed from GPT-5's reasoning process align with human intuition. Together with the strong but unstable performance that LLM shows in Section 3, it will be interesting to ask whether we can *distill some insights from LLM into a deterministic program.* To explore this direction, we develop a simple rule-based index tuner guided by these summarized insights.

The simple tuner constructs one covering index per table by analyzing only the original query plan. A technical problem is to determine the order of *key columns* for each index, as we do not observe a consistent pattern from GPT-5. Our approach follows the implicit column-access order with respect to selective operators in the query plan. For example, if the plan first filters a table on column A, then joins it with another table on column B, and the final output is ordered by column C, we construct an index with key columns ordered as (A, B, C). Any additional referenced columns from the same table (e.g., those needed by projection) are incorporated into the index as *included columns*. This approach is motivated by two considerations. First, the index thus generated is likely to be used to improve the current plan, which is the best plan returned by the query optimizer with existing indexes. Second, the new index is less likely to significantly change the current join order, reducing the risk of selecting a new but potentially disastrous one [75].

Algorithm 1 presents the details of our rule-based index tuner. It begins by enumerating all base tables referenced in the plan and initializing three variables for each table: (1) the ordered key-column list K, (2) the set of referenced columns A, and (3) the cumulative access cost C (lines 2 to 3). It then performs a depth-first traversal of the query plan via the procedure Traverse, which updates A and C whenever a Scan operator is encountered (lines 14 to 16), and appends columns to K when they appear in selective or ordering operators, e.g., IndexSeek, Filter, Join, GroupBy, OrderBy (lines 17 to 18). After traversing the query plan, it constructs a covering index for each table whose cumulative access cost is significant enough, using the collected list of key columns and the set of referenced columns (lines 6 to 8). To determine the significance of the cumulative access cost of a table, we specify a percentage threshold $0 \leq \alpha < 1$. A table is costly enough if its cumulative access cost is greater than α timed by the total plan cost (line 7).



(a) **TPC-H** $\alpha = 0$ (b) **Real-D** $\alpha = 0$ (c) **Real-M** $\alpha = 0$ (d) **Real-R** $\alpha = 0$ (e) **Real-S** $\alpha = 0$

(f) **TPC-H** $\alpha = 0.01$ (g) **Real-D** $\alpha = 0.01$ (h) **Real-M** $\alpha = 0.01$ (i) **Real-R** $\alpha = 0.01$ (j) **Real-S** $\alpha = 0.01$

Figure 8: Evaluation of the simple index tuner (y-axis: ■ DTA time, ▲ best time of DTA and the simple index tuner).

Algorithm 1: `SimpleIndexRecommendation`(r, α)

Input: r: root of the plan tree; α: index-building threshold.
Output: $\mathcal{I}$: set of index recommendations.

$\mathcal{T} \leftarrow$ `getAllBaseTables`(r);
foreach $t \in \mathcal{T}$ **do**
  $K[t] \leftarrow [\,]$; $A[t] \leftarrow \emptyset$; $C[t] \leftarrow 0$;
`Traverse`(r, K, A, C);
$\mathcal{I} \leftarrow \emptyset$;
foreach $t \in \mathcal{T}$ **do**
  if $C[t] > \alpha \cdot r.\text{cost}$ **then**
    // Build a covering index.
    $\mathcal{I} \leftarrow \mathcal{I} \cup$ `buildIndex`(table = t, keyCols = $K[t]$, includedCols = $A[t] \setminus K[t]$);
return $\mathcal{I}$;

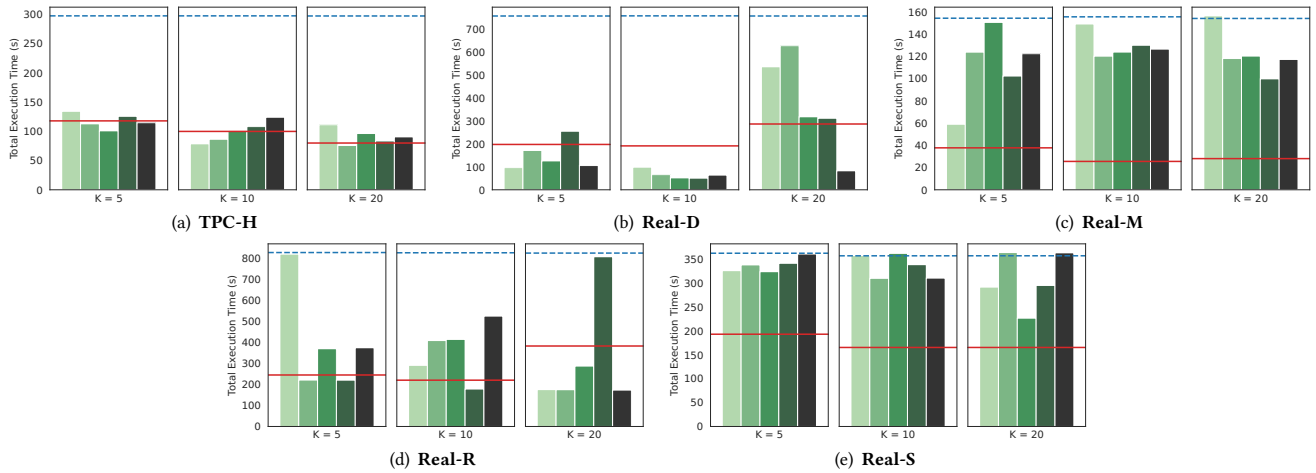
Traverse (n, K, A, C)
  foreach $n_c \in n$.children **do**
    `Traverse`(n_c, K, A, C);
  if n.type = *Scan* **then**
    // Accumulate base table access cost.
    $C[n.\text{table}] \leftarrow C[n.\text{table}] + n.\text{cost}$;
    // Collect all referenced columns.
    $A[n.\text{table}] \leftarrow A[n.\text{table}] \cup n.\text{allRefCols}$;
  foreach $c \in n.\text{seekCols} \cup n.\text{predicateCols} \cup n.\text{groupByCols} \cup n.\text{orderByCols} \cup n.\text{joinKeyCols}$ **do**
    // Collect key columns in order of first access.
    $K[c.\text{table}] \leftarrow$ `appendIfNotExists`$(K[c.\text{table}], c)$;

4.3 Evaluation of the Rule-based Index Tuner

Figure 8 presents the evaluation results of the rule-based index tuner (Algorithm 1), with all five workloads using $\alpha = 0$ (i.e., building indexes for all tables) and $\alpha = 0.01$ (i.e., skipping tables whose estimated access cost $\leq 1\%$ of the total plan cost). Again, only cases where LLM outperforms DTA are shown. To highlight the complementary benefit that this simple tuner can offer, we report results only for queries where the LLM outperforms DTA.

For each query, we compare the execution time obtained under different tuning strategies. The x-axis shows the best execution time achieved by LLM among its five responses. The y-axis reports the execution time under two interpretations, distinguished by the shape of the marker: square markers denote the execution time under DTA, while triangle markers denote the better execution time between DTA and the simple tuner. The latter reflects the best possible performance when combining the recommendations of both DTA and the simple rule-based tuner. The dashed line indicates equal performance on both axes.

- **TPC-H**: There are four queries where the best among the five LLM responses substantially outperforms DTA, as indicated by the four blue squares far above the diagonal line. In all of these cases, the simple rule-based tuner (with both $\alpha = 0$ and 0.01) identifies recommendations that are comparable to those of the LLM, with all red triangles lying close to the diagonal.
- **Real-D**: With $\alpha = 0$, the simple rule-based tuner effectively reduces the number of queries for which DTA performs substantially worse than LLM. In particular, two queries that originally timed out (i.e., exceeding the five-minute limit) or nearly timed out after tuning by both LLM and DTA can now be completed in about ten seconds, as indicated by the two rightmost pairs of markers. In contrast, skipping indexes on low-cost tables (with $\alpha = 0.01$) does not improve these cases, indicating that even small tables can matter from an indexing perspective. Aside from these two queries, the simple tuner yields similar improvements across different threshold values. However, there are still queries where the LLM finds significantly better configurations even after incorporating the rule-based tuner, as reflected by triangles above the diagonal line.
- **Real-M**: The rule-based tuner substantially narrows the gap between DTA and LLM for queries with the largest performance differences (note that the execution time is shown with a logarithmic scale). However, it does not produce comparable recommendations in all cases. The performance of the rule-based tuner is also insensitive to the choice of threshold α.
- **Real-R**: The observation is similar to that with **Real-M**, as the rule-based tuner (with both $\alpha = 0$ and 0.01) can improve DTA and narrow its gap with LLM. Nevertheless, not all queries can be improved to match the performance of LLM.
- **Real-S**: There is only one query for which LLM substantially outperforms DTA. Unfortunately, the rule-based tuner cannot identify recommendations that improve DTA for this case.



(a) **TPC-H** (b) **Real-D** (c) **Real-M**

(d) **Real-R** (e) **Real-S**

Figure 9: LLM-driven index tuning (five invocations shown as bars) vs. DTA (red line) for tuning multi-query workloads.

Key Finding 6. *In cases where DTA underperforms LLM, the simple rule-based index tuner produces better recommendations than DTA in approximately 60% of our test queries, and achieves performance comparable to or better than LLM in about 40% of all test cases.*

This observation is encouraging, as it suggests that, by distilling insights from LLM, we can achieve substantial improvements in many cases by only validating two sets of recommendations: those from DTA and those from the simple rule-based index tuner.

Remarks. The simple tuner serves as a proof of concept demonstrating that effective heuristics can be distilled from LLM without inheriting the risks of high variance in index recommendation quality. We observe that this simple rule-based tuner is effective on its own, particularly in cases where DTA's recommendations underperform. A more comprehensive evaluation of the rule-based tuner is included in the full version of this paper [67].

5 MULTI-QUERY WORKLOADS

We extend our study to multi-query workloads, a more common setting in industrial practice. Unlike tuning a single query, tuning a multi-query workload calls for the index tuner to disregard indexes that are promising locally (for some queries) but unfavorable globally (for the entire workload). Practical constraints, such as the number of indexes or storage space limits, further restrict the ability to materialize all beneficial indexes.

5.1 Overview of Evaluation Results

We construct multi-query workloads as follows. For **TPC-H**, **Real-R**, and **Real-S**, we include all queries in a single workload. For **Real-D** and **Real-M**, whose queries are more complex and exceed GPT-5's context window when considered jointly, we select the 10 queries with the largest observed improvements under single-query tuning by GPT-5 (as presented in Section 3).

Figure 9 presents the results for five invocations on all multi-query workloads, with the blue dashed line denoting the workload execution time with the original index configuration. The solid red line represents the workload execution time with the indexes recommended by DTA. The average response time of GPT-5 is 4.4, 3.8, 2.3, 2.4, and 3.4 minutes for **TPC-H**, **Real-M**, **Real-D**, **Real-R**, and **Real-S**, respectively. We vary the constraint K on the number of indexes allowed as $K \in \{5, 10, 20\}$. We do not observe violation of these constraints from the responses of LLM in our evaluation.

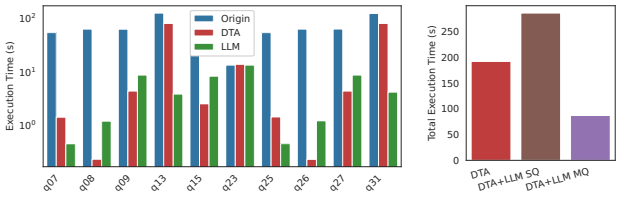
- **TPC-H**: The performance of LLM is similar to that of DTA, regardless of the maximum number of indexes allowed K.
- **Real-D**: With $K = 5$ or $K = 10$, LLM-recommended indexes yield lower execution time than DTA-recommended indexes. However, when $K = 20$, we observe significant performance degradation in most LLM invocations, a trend also observed for DTA. This is surprising because increasing the number of indexes should in general reduce workload execution time. Moreover, we observe a large performance variation with configurations recommended by different LLM invocations.
- **Real-M**: For all constraints $K \in \{5, 10, 20\}$ tested, LLM-driven index tuning leads to significantly longer workload execution times compared to DTA.
- **Real-R**: LLM-driven index tuning exhibits substantial performance variation. Although the best responses outperform DTA, the worst responses lead to much longer execution times. With $K = 5$ or $K = 20$, the worst LLM outcomes achieve almost no improvement over the original configuration.
- **Real-S**: Of all the workloads evaluated, LLM performs the worst for **Real-S**. Most LLM responses yield little improvement over the original index configuration, with only a few exceptions. In contrast, DTA achieves substantially better performance.

Key Finding 7. *Overall, DTA provides more stable and reliable improvements for multi-query workloads while LLM's performance varies considerably across workloads and invocations. Nevertheless, there are cases in which LLM can significantly outperform DTA.*

This observation is based on workloads constructed using the top 10 queries with the largest single-query improvements using LLM for **Real-D** and **Real-M**. Therefore, including other less beneficial queries will further reduce the relative effectiveness of LLM. We next present two case studies to examine the factors behind LLM's successes and failures on multi-query workloads.

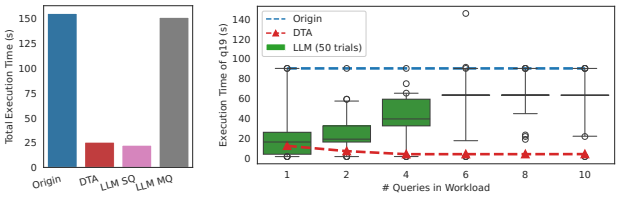
5.2 Case Study: What Went Right for Real-D?

We first take a closer look at **Real-D** with $K = 10$. In this setting, GPT-5 delivers exceptionally strong performance, where all five invocations substantially outperform DTA, with the best achieving



(a) Execution time of all queries. (b) Variations of DTA.

Figure 10: Analysis of Real-D $K = 10$.



(a) Variations of LLM. (b) Impact of workload size to dominant query tuning.

Figure 11: Analysis of Real-M $K = 10$.

4× further speedup. For the remainder of our analysis, we use this best-performing trial as a representative result for LLM.

Is the Comparison Fair? Because we constrain the LLM by the number of indexes rather than by storage space, a natural question is whether its performance gains arise from generating disproportionately large indexes. To examine this, we measured the storage footprint of the index recommendations produced by both systems. We found that DTA's indexes occupy 13.7 GB of storage, while LLM's indexes occupy only 7.7 GB. This shows that LLM does not achieve its performance by inflating the storage, indicating that its recommendation is indeed superior in this case.

Per-query Improvement. Beyond total execution time, we also evaluated the benefit of the index recommendations at the per-query level. Figure 10(a) reports the execution time of each query before and after tuning by both DTA and LLM. We observe that nine out of the ten queries exhibit substantial speedups under both methods. Among these nine queries, LLM outperforms DTA for four queries, while DTA performs better for the remaining five queries. This indicates that both approaches optimize the workload effectively, with each favoring different subsets of queries.

Cross-query Recommendation. To better understand GPT-5's behavior in this case, we examine its underlying reasoning process. Rather than selecting the most promising indexes for each query in isolation, GPT-5 appears to prioritize indexes that can benefit multiple queries. The following excerpt illustrates this behavior (all table and column names are anonymized). We observe similar reasoning patterns across all multi-query workloads.

> *... Indexing Tab_1 on (Col_2, Col_3, Col_4, Col_1) including Col_5 could benefit multiple queries, especially Q_1, Q_4 and Q_5 ... I need to limit the proposals to 10 indexes, considering one that aids both queries on Tab_2 ...*

To assess the effectiveness of this intention, we analyze the contribution of each index in LLM's recommendation. Specifically, for every index, we count how many queries actually use it. We find that the top three most frequently used indexes appear in 9, 7, and 5 of the 10 queries, respectively. For comparison, the top three indexes recommended by DTA are used by 5, 4, and 3 queries.

Improve DTA with LLM. Furthermore, we find that none of the three most beneficial LLM-recommended indexes is generated as candidates by DTA. This observation raises an interesting question: can LLM-recommended indexes enrich the candidate pool of DTA and thus improve its tuning performance? To investigate this, we modify DTA's candidate index generation process to incorporate LLM's recommendations. Figure 10(b) presents the results, where "DTA+LLM SQ" and "DTA+LLM MQ" are two variants that incorporate LLM's recommendations for each individual query (as in Section 3) and for the entire workload, respectively. The result shows that LLM's multi-query workload recommendations improve DTA's performance by more than 2×. However, including only LLM's single-query recommendations fails to improve performance and, in fact, significantly slows down overall execution time. This degradation again results from inaccurate cost estimates.

Key Finding 8. *Given a multi-query workload, LLM tends to recommend indexes that can benefit multiple queries, which may produce high-quality candidate indexes that DTA misses.*

This finding points to an opportunity to improve DTA by expanding its candidate pool with LLM-recommended indexes. However, as we show in Section 6.1, directly incorporating such recommendations often leads to performance degradation. Nevertheless, this highlights the potential for integrating LLM's insights into DTA if cost estimation can be further improved, an area that continues to receive active research attention.

5.3 Case Study: What Went Wrong for Real-M?

Next, we study the case of **Real-M** with $K = 10$. As shown in Figure 9(c), all five LLM invocations consistently fail to achieve improvements comparable to those of DTA.

Recommendation Analysis. Unlike **Real-D** where all queries (without tuning) have relatively similar execution time (Figure 10(a)), **Real-M** is dominated by two queries (i.e., queries 19 and 22), together accounting for roughly 85% of the total execution time (60% and 25%, respectively). Consequently, effective workload optimization hinges on whether LLM can identify these bottleneck queries and recommend indexes to improve them. The five invocations of GPT-5 clearly fail in this regard. Among the five responses, only one proposes indexes that can be used by query 19, and only two have impact on query 22. Moreover, these responses fail to target the real bottlenecks. In contrast, nearly 90% of DTA's improvement for the entire workload comes from these two queries.

We further compare the quality of LLM's multi-query recommendations with those produced when each query is provided in isolation (as in Section 3). Specifically, we collect all the indexes recommended by LLM from five multi-query responses and, separately, five single-query responses, forming two pools of candidates. To obtain a final recommendation within the constraint, for each pool of candidates, we use DTA's configuration enumeration algorithm (excluding DTA's own candidate indexes) to select the best 10 indexes. Figure 11(a) reports the results, where configurations derived from the multi-query and single-query settings are denoted as "LLM MQ" and "LLM SQ," respectively. We can see that while LLM MQ barely improves workload performance, LLM SQ achieves more than 80% improvement, reaching a level comparable to DTA.

Explaining LLM's Poor Performance. The above observation suggests that instead of focusing on the most costly queries, the LLM attempts to identify patterns that improve the workload as a whole and, as a result, overlooks the actual bottlenecks.

Table 3: Performance impact on DTA from including LLM-recommended indexes as additional candidates.

	Single-query (# queries)			Multi-query
Workload	Improved	Unchanged	Degraded	Improvement
TPC-H	6	12	4	15.37%
Real-D	10	8	11	54.64%
Real-M	8	18	14	1.29%
Real-R	6	9	9	-87.33%
Real-S	0	6	6	-71.06%

To examine this hypothesis, we vary the workload size and measure its impact on the most costly query (i.e., query 19). Starting from a single-query workload containing only query 19, we incrementally add additional queries to create workloads of sizes {2, 4, 6, 8, 10}. For each workload, we collect 50 LLM responses to analyze the distribution of their impacts on query 19. Figure 11(b) summarizes the results, with whiskers indicating the 5th and 95th percentiles. For reference, the figure also includes the execution time of query 19 without tuning and with DTA tuning, denoted by the blue and red dashed lines, respectively.

When the workload contains only query 19, roughly 50% of LLM invocations achieve performance comparable to or better than DTA. This fraction drops sharply by adding only one more query, and falls below 5% once the workload includes four queries. The median improvement of query 19 increases monotonically as more queries are added, stabilizing at a workload size of six. This result indicates that LLM quality degrades when more queries are included, despite the fact that the estimated costs in the prompt encode the relative importance of each query. In contrast, under the cost-based tuning framework, DTA's improvement for query 19 remains stable and even improves slightly as the workload grows, perhaps due to additional statistics created while tuning other queries.

Key Finding 9. *When information of multiple queries are provided in a plain, concatenated form without additional selection or processing, LLM places less emphasis on the most costly queries leading to suboptimal index recommendations.*

Note that the above behavior is not unique to index tuning or to the GPT-5 model. Previous work reports similar tendencies in various LLMs when exposed to irrelevant contexts [57], or long contexts that are relevant but overly extensive [22], across a range of NLP tasks. Our result shows that such degradation in LLM quality with a larger number of queries can become a serious issue when the underlying workload is complex and unbalanced. Existing work on workload compression [7, 58] and cost-based prompt construction [25, 84] potentially offers alternatives for mitigating this issue (see Sections 7 and 8).

6 LESSONS FROM LLM-DTA INTEGRATION

We show that it is nontrivial to integrate LLM with DTA, with two attempts. For simplicity, we focus on directly integrating LLM-recommended indexes (Sections 3 and 5) with DTA. Integrating LLM-derived insights (Section 4) leads to similar challenges.

6.1 Cost-based Integration

One plausible way to leverage LLM's capability is to integrate its recommendations into the existing cost-based framework of DTA. An implementation of this approach has been demonstrated in Section 5.2 with the **Real-D** workload, using LLM to only enrich DTA's pool of candidate indexes while keeping the other components of DTA unaffected. This integration preserves DTA's contract to select the configuration with the lowest estimated cost, while expanding its search space with LLM-recommended indexes.

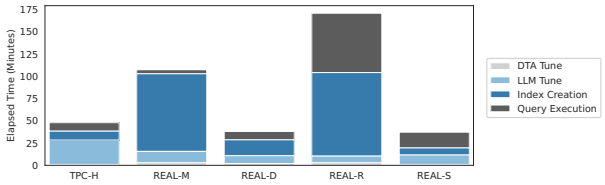


Figure 12: Time breakdown for performance validation.

Table 3 summarizes the evaluation results of this implementation for all workloads, comparing the final configurations found by DTA with and without the LLM-recommended indexes. For single-query workloads, we report the number of queries whose execution time improves by at least 5% ("Improved"), remains within ±5% ("Unchanged"), or degrades by at least 5% ("Degraded"). For multi-query workloads, we report the overall performance change measured by the total execution time. We have the following finding.

Key Finding 10. *Enriching DTA's pool of candidates with LLM-recommended indexes does not consistently improve performance, and often leads to performance degradation.*

This frequent performance degradation can again be attributed to inaccurate cost estimates. Despite decades of progress made, cost estimation remains a challenging problem with active research. Advances in this direction may help mitigate the issue if adopted in production. Here, we emphasize the practical impact of cost estimation errors (stemming from inaccurate cardinality estimates) in the context of integrating LLM with DTA. As a production-grade tuner, DTA serves as a substantially strong starting point and is therefore more sensitive to inaccuracies in cost estimation.

6.2 Validation-based Integration

Another approach is to incorporate performance validation. One can materialize the recommended configurations (from both DTA and multiple LLM invocations), execute the corresponding queries, and select the configuration with the best execution time.

Figure 12 shows the end-to-end time breakdown of this approach across all multi-query workloads. Specifically, "DTA Tune" and "LLM Tune" report the tuning time of DTA and the total time of the five GPT-5 invocations, respectively. "Index Creation" and "Query Execution" represent the combined index-building and workload-execution time for the configurations recommended by DTA and by the five LLM runs. Different configurations may contain overlapping indexes, and we build each distinct index only once. We first evaluate the recommendation by DTA, followed by the five LLM-generated configurations, using the minimum observed execution time as the timeout for subsequent validations.

Key Finding 11. *The cost of performance validation of recommended configurations is often significantly higher than the cost of index tuning itself, largely due to the overhead of index creation.*

Previous work has identified similar challenges and proposed mitigation techniques that focus primarily on reducing query execution cost [25, 42] or improving data efficiency [73]. Here, we highlight the overhead of index creation, which can become a more

substantial bottleneck and is easily amplified by the diverse recommendations that LLM could produce.

7 FUTURE OPPORTUNITIES

Enhancing LLM-driven Index Tuning. In this paper, we evaluate pre-trained LLM for index tuning in an end-to-end setting, using only basic database and workload information. Beyond this setup, there are several opportunities to further leverage LLM for index tuning. For example, prompts can be enriched with additional information, such as DTA recommended indexes for the same query or similar queries, which can serve as reference demonstrations [84]. In addition, LLM can be adapted to the index tuning task through task-specific fine-tuning [28, 52, 55, 56]. Moreover, the tuning process can be decomposed into multiple tasks and handled by a multi-agent framework [32, 38, 40], where existing index tuners such as DTA can be integrated as an external tool to provide additional guidance and/or feedback at different stages. Understanding how these enhancements impact real-world workloads, particularly in addressing the limitations of LLM observed in our study, such as large performance variance and the tendency toward distraction, remains an interesting direction for future work.

Distilling Insights from LLM. As demonstrated in Section 4, heuristics used by LLM could be distilled and exploited to narrow the gap between existing index tuners and the best of LLM. Whether additional insights can be extracted and applied more systematically remains an open question. Beyond integrating these insights into existing index tuners, an alternative is to distill the knowledge into smaller, task-specific models [27, 47, 80]. Such models may provide a more efficient and predictable option for production deployment while retaining much of the effectiveness of LLM.

Making Performance Validation a First-class Citizen in Index Tuning. A key roadblock in using LLM-recommended indexes with today's cost-based index tuners is the inaccuracy of cost estimation. As a result, incorporating performance validation as a first-class citizen appears increasingly important to production index tuners. Some recent industrial systems [10, 16, 78] have incorporated validation mechanisms, but typically only to verify the final configuration selected by the tuner. This restriction is largely due to the substantial cost of validation. As shown in Section 6.2, incorporating LLM-driven index tuning further amplifies this challenge, as LLM can easily generate a large and diverse set of candidate recommendations, significantly increasing the need for validation. Therefore, an important direction for future work is to develop low-overhead while less disruptive [53] validation techniques. A further step is to integrate this validation component more tightly into the index tuning process, allowing validation outcomes to complement existing cost estimates and directly influence tuning decisions.

8 RELATED WORK

LLM-driven Index Tuning. A flurry of recent work has been proposed to unlock the power of LLMs to improve the performance of database systems [25, 36, 63]. Most of this line of work is devoted to tuning configuration parameters (a.k.a. knobs) of database systems, which itself is a fast-growing research area with intensive studies [4, 5, 9, 23, 34, 39, 65, 83]. To the best of our knowledge, so far there has been little work on using LLM for index tuning yet, except for a couple of very recent proposals, such as λ-Tune [25], LLMIdxAdvis [84], and MAAdvisor [40]. Both λ-Tune and LLMIdxAdvis follow the end-to-end workflow evaluated in our study, while MAAdvisor proposes a different multi-agent framework with a much more complicated workflow. Unlike the classic cost-based index tuning that has been well studied and evaluated in real-world industrial deployments [10, 16, 78], LLM-driven index tuning remains an underexplored area, especially with an industrial setup.

Workload Compression. As we demonstrated in Section 5, LLM-driven index tuning does not have an advantage over DTA when tuning multi-query workloads and workload size does have an impact on the performance of LLM. Therefore, workload compression [7, 11, 18, 60] will continue to be an interesting and critical direction for future work to improve the efficacy of LLM for tuning multi-query workloads, which has also explored by λ-Tune [25]. Recent work on foundation database models [70] proposes learning compact representations of datasets and workloads in a latent space that generalize across tasks, which could also be used to guide workload compression for LLM-based index tuners.

Robustness of LLMs. In addition to database performance tuning (including index tuning), LLMs have also been applied to other aspects of improving database performance, such as performance diagnosis [24, 86] and query optimization [6, 61]. One common issue found by this line of work is the robustness of LLM invocations, which often respond with incorrect or irrelevant results due to problems such as hallucination. We have also highlighted such robustness issues for LLM-driven index tuning in Section 3.4, which calls for future research effort. One solution to improve robustness, as was proposed by λ-Tune, is to include a postprocessing validation stage that creates the recommended indexes by LLM and executes the workload queries to identify the best indexes. Although this solution is guaranteed to find the best indexes among the recommended candidates, it incurs considerable overhead that is perhaps infeasible in practical industrial applications, as shown in Section 6.1. Therefore, the development of less costly validation technologies is another area of interest for future work.

9 CONCLUSION

We empirically evaluated the effectiveness of LLM-driven index tuning on real-world and industry benchmark workloads on Microsoft SQL Server, and compared it with a commercial index tuner, Database Tuning Advisor (DTA). Our results reveal that LLMs demonstrate the potential to provide high-quality index recommendations, particularly in cases where DTA's recommendations are not effective due to errors in the query optimizer's cost estimation. However, LLMs exhibit substantial variance in the quality of index recommendations, both across invocations for the same query and across queries. Furthermore, incorporating LLM-recommended indexes as additional candidates in DTA's search space yielded no significant improvements in quality. Thus, despite their promise, the use of LLMs for index tuning in practice remains an open and challenging problem. Based on our findings, we identify potential areas of future work that aim to make LLM-driven index tuning more practical, such as techniques that can reduce the variance of LLMs without significantly reducing its quality, as well as more efficient mechanisms for validating index recommendations.

REFERENCES

[1] 2025. Microsoft SQL Server Missing Indexes. https://learn.microsoft.com/en-us/sql/relational-databases/indexes/tune-nonclustered-missing-index-suggestions?view=sql-server-ver17.

[2] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollár, Arunprasad P. Marathe, Vivek R. Narasayya, and Manoj Syamala. 2004. Database Tuning Advisor for Microsoft SQL Server 2005. In *VLDB*. 1110–1121.

[3] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. 2001. Materialized View and Index Selection Tool for Microsoft SQL Server 2000. In *SIGMOD*.

[4] Dana Van Aken et al. 2021. An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems. *Proc. VLDB Endow.* 14, 7 (2021), 1241–1253.

[5] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *SIGMOD*. ACM, 1009–1024.

[6] Peter Akioyamen, Zixuan Yi, and Ryan Marcus. 2024. The Unreasonable Effectiveness of LLMs for Query Optimization. *CoRR* abs/2411.02862 (2024).

[7] Matteo Brucato, Tarique Siddiqui, Wentao Wu, Vivek Narasayya, and Surajit Chaudhuri. 2024. Wred: Workload Reduction for Scalable Index Tuning. *Proc. ACM Manag. Data* 2, 1, Article 50 (2024), 26 pages.

[8] Nicolas Bruno and Surajit Chaudhuri. 2005. Automatic Physical Database Tuning: A Relaxation-based Approach. In *SIGMOD*. 227–238.

[9] Stefano Cereda et al. 2021. CGPTuner: a Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations Under Varying Workload Conditions. *Proc. VLDB Endow.* 14, 8 (2021), 1401–1413.

[10] Sunil Chakkappen et al. 2025. Automatic Indexing in Oracle. *Proc. VLDB Endow.* 18, 12 (2025), 4924–4937.

[11] Surajit Chaudhuri, Ashish Kumar Gupta, and Vivek R. Narasayya. 2002. Compressing SQL workloads. In *SIGMOD*. 488–499.

[12] Surajit Chaudhuri and Vivek Narasayya. 2020. Anytime Algorithm of Database Tuning Advisor for Microsoft SQL Server.

[13] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *VLDB*. 146–155.

[14] Surajit Chaudhuri and Vivek R. Narasayya. 1998. AutoAdmin 'What-if' Index Analysis Utility. In *SIGMOD*. 367–378.

[15] Surajit Chaudhuri and Vivek R. Narasayya. 1999. Index Merging. In *ICDE*.

[16] Sudipto Das et al. 2019. Automatically Indexing Millions of Databases in Microsoft Azure SQL Database. In *SIGMOD*. 666–679.

[17] Debabrata Dash, Neoklis Polyzotis, and Anastasia Ailamaki. 2011. CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads. *Proc. VLDB Endow.* 4, 6 (2011), 362–372.

[18] Shaleen Deep, Anja Gruenheid, Paraschos Koutris, Jeffrey F. Naughton, and Stratis Viglas. 2020. Comprehensive and Efficient Workload Compression. *Proc. VLDB Endow.* 14, 3 (2020), 418–430.

[19] Sriram Dharwada, Himanshu Devrani, Jayant Haritsa, and Harish Doraiswamy. 2025. Query rewriting via llms. *arXiv preprint arXiv:2502.12918* (2025).

[20] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2019. AI Meets AI: Leveraging Query Executions to Improve Index Recommendations. In *SIGMOD*. 1241–1258.

[21] Bailu Ding, Sudipto Das, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2018. Plan Stitch: Harnessing the Best of Many Plans. *Proc. VLDB Endow.* 11, 10 (2018), 1123–1136.

[22] Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Babu Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. 2025. Context Length Alone Hurts LLM Performance Despite Perfect Retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 23281–23298. https://doi.org/10.18653/v1/2025.findings-emnlp.1264

[23] Songyun Duan et al. 2009. Tuning Database Configuration Parameters with iTuned. *Proc. VLDB Endow.* 2, 1 (2009), 1246–1257.

[24] Victor Giannakouris and Immanuel Trummer. 2024. DBG-PT: A Large Language Model Assisted Query Performance Regression Debugger. *Proc. VLDB Endow.* 17, 12 (2024), 4337–4340. https://www.vldb.org/pvldb/vol17/p4337-giannakouris.pdf

[25] Victor Giannakouris and Immanuel Trummer. 2025. λ-Tune: Harnessing Large Language Models for Automated Database System Tuning. *Proc. ACM Manag. Data* 3, 1 (2025), 2:1–2:26.

[26] Goetz Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.* 18, 3 (1995), 19–29.

[27] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. MiniLLM: Knowledge Distillation of Large Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=5h0qf7IBZZ

[28] Beliz Gunel, Jingfei Du, Alexis Conneau, and Veselin Stoyanov. 2021. Supervised Contrastive Learning for Pre-trained Language Model Fine-tuning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=cu7IUiOhujH

[29] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[30] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).

[31] Yannis E. Ioannidis and Stavros Christodoulakis. 1991. On the Propagation of Errors in the Size of Join Results. In *SIGMOD*. 268–277.

[32] Kunal Jha, Wilka Carvalho, Yancheng Liang, Simon Shaolei Du, Max Kleiman-Weiner, and Natasha Jaques. 2025. Cross-environment Cooperation Enables Zero-shot Multi-agent Coordination. In *ICML*.

[33] Andrew Kane. 2017. Introducing Dexter, the Automatic Indexer for Postgres. https://medium.com/@ankane/introducing-dexter-the-automatic-indexer-for-postgres-5f8fa8b28f27.

[34] Konstantinos Kanellis et al. 2022. LlamaTune: Sample-Efficient DBMS Configuration Tuning. *Proc. VLDB Endow.* 15, 11 (2022), 2953–2965.

[35] Jan Kossmann, Stefan Halfpap, Marcel Jankrift, and Rainer Schlosser. 2020. Magic mirror in my hand, which is the best in the land? An Experimental Evaluation of Index Selection Algorithms. *Proc. VLDB Endow.* 13, 11 (2020), 2382–2395.

[36] Jiale Lao et al. 2025. GPTuner: An LLM-Based Database Tuning System. *SIGMOD Rec.* 54, 1 (2025), 101–110.

[37] Viktor Leis et al. 2015. How Good Are Query Optimizers, Really? *PVLDB* 9, 3 (2015), 204–215.

[38] Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. 2025. Agent-Oriented Planning in Multi-Agent Systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. https://openreview.net/forum?id=EqcLAU6gyU

[39] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. *Proc. VLDB Endow.* 12, 12 (2019), 2118–2130.

[40] Zhaodonghui Li, Haitao Yuan, Jiachen Shi, Hao Zhang, Yu Rong, and Gao Cong. 2025. MAAdvisor: Zero-Shot Index Advisor using Multi-Agent LLMs. *CoRR* abs/2508.16044 (2025).

[41] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024. LLM-R2: A Large Language Model Enhanced Rule-Based Rewrite System for Boosting Query Efficiency. *Proc. VLDB Endow.* 18, 1 (Sept. 2024), 53–65. https://doi.org/10.14778/3696435.3696440

[42] Wan Shen Lim, Lin Ma, William Zhang, Matthew Butrovich, Samuel Arch, and Andrew Pavlo. 2024. Hit the gym: accelerating query execution to efficiently bootstrap behavior models for self-driving database management systems. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3680–3693.

[43] Jie Liu and Barzan Mozafari. 2024. GenRewrite: Query Rewriting via Large Language Models. *arXiv preprint arXiv:2403.09060* (2024).

[44] Guy Lohman. [n.d.]. Is Query Optimization a "Solved" Problem? http://wp.sigmod.org/?p=1075.

[45] Lin Ma, Bailu Ding, Sudipto Das, and Adith Swaminathan. 2020. Active Learning for ML Enhanced Database Systems. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 175–191. https://doi.org/10.1145/3318464.3389768

[46] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query Rewriting in Retrieval-Augmented Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 5303–5315. https://doi.org/10.18653/v1/2023.emnlp-main.322

[47] Amir M. Mansourian, Rozhan Ahmadi, Masoud Ghafouri, Amir Mohammad Babaei, Elaheh Badali Golezani, Zeynab yasamani ghamchi, Vida Ramezanian, Alireza Taherian, Kimia Dinashi, Amirali Miri, and Shohreh Kasaei. 2025. A Comprehensive Survey on Knowledge Distillation. *Transactions on Machine Learning Research* (2025). https://openreview.net/forum?id=3cbJzdR78B

[48] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *Proc. VLDB Endow.* 12, 11 (2019), 1733–1746. https://doi.org/10.14778/3342263.3342646

[49] Microsoft. 2026. Azure Virtual Machines. https://azure.microsoft.com/en-us/products/virtual-machines.

[50] Vivek R. Narasayya and Surajit Chaudhuri. 2026. Leveraging Query Optimizers to Verify the Soundness of LLM-based Query Rewrites for Real-World Workloads, and More. In *16th Conference on Innovative Data Systems Research, CIDR 2026, Chaminade, CA, USA, January 18-21, 2026*. www.cidrdb.org. https://vldb.org/cidrdb/2026/leveraging-query-optimizers-to-verify-the-soundness-of-llm-based-query-rewrites-for-real-world-workloads.html

[51] OpenAI. 2025. Gpt-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf.

[52] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your

Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html

[53] Neon Savannah Longoria. 2025. Practical Guide to Database Branching. https://neon.com/blog/practical-guide-to-database-branching.

[54] Rainer Schlosser, Jan Kossmann, and Martin Boissier. 2019. Efficient Scalable Multi-attribute Index Selection Using Recursive Strategies. In *ICDE*. 1238–1249.

[55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

[56] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *CoRR* abs/2402.03300 (2024). https://doi.org/10.48550/ARXIV.2402.03300 arXiv:2402.03300

[57] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. 2023. Large Language Models Can Be Easily Distracted by Irrelevant Context. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.), Vol. 202. PMLR, 31210–31227. https://proceedings.mlr.press/v202/shi23a.html

[58] Tarique Siddiqui, Saehan Jo, Wentao Wu, Chi Wang, Vivek R. Narasayya, and Surajit Chaudhuri. 2022. ISUM: Efficiently Compressing Large and Complex Workloads for Scalable Index Tuning. In *SIGMOD*. 660–673.

[59] Tarique Siddiqui and Wentao Wu. 2023. ML-Powered Index Tuning: An Overview of Recent Progress and Open Challenges. *SIGMOD Rec.* 52, 4 (2023), 19–30.

[60] Tarique Siddiqui, Wentao Wu, Vivek R. Narasayya, and Surajit Chaudhuri. 2022. DISTILL: Low-Overhead Data-Driven Techniques for Filtering and Costing Indexes for Scalable Index Tuning. *Proc. VLDB Endow.* 15, 10 (2022), 2019–2031.

[61] Zhaoyan Sun, Xuanhe Zhou, Guoliang Li, Xiang Yu, Jianhua Feng, and Yong Zhang. 2025. R-Bot: An LLM-based Query Rewrite System. *Proc. VLDB Endow.* 18, 12 (2025), 5031–5044.

[62] Jie Tan, Kangfei Zhao, Rui Li, Jeffrey Xu Yu, Chengzhi Piao, Hong Cheng, Helen Meng, Deli Zhao, and Yu Rong. 2025. Can Large Language Models Be Query Optimizer for Relational Databases? *Proc. ACM Manag. Data* 3, 6 (2025), 1–28. https://doi.org/10.1145/3769771

[63] Immanuel Trummer. 2024. DB-BERT: making database tuning tools "read" the manual. *VLDB J.* 33, 4 (2024), 1085–1104.

[64] Gary Valentin et al. 2000. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In *ICDE*. 101–110.

[65] Junxiong Wang et al. 2021. UDO: Universal Database Optimization using Reinforcement Learning. *Proc. VLDB Endow.* 14, 13 (2021), 3402–3414.

[66] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *Proc. VLDB Endow.* 14, 9 (2021), 1640–1654.

[67] Xiaoying Wang, Wentao Wu, Vivek Narasayya, and Surajit Chaudhuri. 2026. Evaluating the Practical Effectiveness of LLM-Driven Index Tuning on Microsoft SQL Server. https://arxiv.org/pdf/2603.09181.

[68] Xiaoying Wang, Wentao Wu, Vivek R. Narasayya, and Surajit Chaudhuri. 2025. Esc: An Early-Stopping Checker for Budget-aware Index Tuning. *Proc. VLDB Endow.* 18, 5 (2025), 1278–1290.

[69] Xiaoying Wang, Wentao Wu, Chi Wang, Vivek R. Narasayya, and Surajit Chaudhuri. 2024. Wii: Dynamic Budget Reallocation In Index Tuning. *Proc. ACM Manag. Data* 2, 3 (2024), 182.

[70] Johannes Wehrstein, Carsten Binnig, Fatma Özcan, Shobha Vasudevan, Yu Gan, and Yawen Wang. 2025. Towards Foundation Database Models. In *15th Conference on Innovative Data Systems Research, CIDR 2025, Amsterdam, The Netherlands, January 19-22, 2025*. www.cidrdb.org. https://vldb.org/cidrdb/2025/towards-foundation-database-models.html

[71] Kyu-Young Whang. 1985. Index Selection in Relational Databases. In *Foundations of Data Organization*. 487–500.

[72] Wentao Wu. 2025. Hybrid Cost Modeling for Reducing Query Performance Regression in Index Tuning. *IEEE Trans. Knowl. Data Eng.* 37, 1 (2025), 379–391. https://doi.org/10.1109/TKDE.2024.3484954

[73] Wentao Wu. 2025. Hybrid Cost Modeling for Reducing Query Performance Regression in Index Tuning. *IEEE Trans. Knowl. Data Eng.* 37, 1 (2025), 379–391.

[74] Wentao Wu, Yun Chi, Shenghuo Zhu, Jun'ichi Tatemura, Hakan Hacigümüs, and Jeffrey F. Naughton. 2013. Predicting query execution time: Are optimizer cost models really unusable?. In *ICDE*. 1081–1092.

[75] Wentao Wu, Anshuman Dutt, Gaoxiang Xu, Vivek R. Narasayya, and Surajit Chaudhuri. 2025. Understanding and Detecting Query Performance Regression in Practical Index Tuning: [Experiments & Analysis]. *Proc. ACM Manag. Data* 3, 6 (2025), 1–26.

[76] Wentao Wu, Jeffrey F. Naughton, and Harneet Singh. 2016. Sampling-Based Query Re-Optimization. In *SIGMOD*. 1721–1736.

[77] Wentao Wu, Chi Wang, Tarique Siddiqui, Junxiong Wang, Vivek R. Narasayya, Surajit Chaudhuri, and Philip A. Bernstein. 2022. Budget-aware Index Tuning with Reinforcement Learning. In *SIGMOD*. 1528–1541.

[78] Ritwik Yadav, Satyanarayana R. Valluri, and Mohamed Zaït. 2023. AIM: A practical approach to automated index management for SQL databases. In *ICDE*. 3349–3362.

[79] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[80] Chuanpeng Yang, Yao Zhu, Wang Lu, Yidong Wang, Qian Chen, Chenlong Gao, Bingjie Yan, and Yiqiang Chen. 2025. Survey on knowledge distillation for large language models: methods, evaluation, and application. *ACM Transactions on Intelligent Systems and Technology* 16, 6 (2025), 1–27.

[81] Zhiming Yao, Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2025. A query optimization method utilizing large language models. *arXiv preprint arXiv:2503.06902* (2025).

[82] Fanghua Ye, Meng Fang, Shenghui Li, and Emine Yilmaz. 2023. Enhancing Conversational Search: Large Language Model-Aided Informative Query Rewriting. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 5985–6006. https://doi.org/10.18653/v1/2023.findings-emnlp.398

[83] Ji Zhang et al. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *SIGMOD*. 415–432.

[84] Xinxin Zhao, Haoyang Li, Jing Zhang, Xinmei Huang, Tieying Zhang, Jianjun Chen, Rui Shi, Cuiping Li, and Hong Chen. 2025. LLMIdxAdvis: Resource-Efficient Index Advisor Utilizing Large Language Model. *CoRR* abs/2503.07884 (2025).

[85] Wei Zhou, Chen Lin, Xuanhe Zhou, and Guoliang Li. 2024. Breaking It Down: An In-Depth Study of Index Advisors. *Proc. VLDB Endow.* 17, 10 (June 2024), 2405–2418. https://doi.org/10.14778/3675034.3675035

[86] Xuanhe Zhou et al. 2024. D-Bot: Database Diagnosis System using Large Language Models. *Proc. VLDB Endow.* 17, 10 (2024), 2514–2527.