LLM4IA: Index Advising via Large Language Models

Xian Lyu
Xiamen University
Xiamen, China
xianlyu2023@stu.xmu.edu.cn

Junbiao Zhang
Xiamen University
Xiamen, China
zhangjunbiao@stu.xmu.edu.cn

Yihang Zheng
Xiamen University
Xiamen, China
yihang@stu.xmu.edu.cn

Guoliang Li
Tsinghua University
Beijing, China
liguoliang@tsinghua.edu.cn

Chen Lin*
Xiamen University
Xiamen, China
chenlin@xmu.edu.cn

Abstract

Recently, large-language models (LLMs) have demonstrated strong potential to solve database problems. However, LLMs still face two challenges in solving the index selection problem: (1) representing the workload in an LLM-friendly form and (2) finding the optimal index set. To solve these challenges, we propose `LLM4IA`, an LLM-based index selection method that can recommend indexes for any analytical workload directly on any database instance. `LLM4IA` proposes a concise description of natural language by extracting and sorting predicates and completely avoiding numerical input. `LLM4IA` adopts an iterative index selection process by repeatedly improving previous index candidates and summarizing effective candidates. Experiments on TPC-H and TPC-DS show that `LLM4IA` surpasses the near-optimal index advisor Extend by 5%-10%. Our demonstration highlights how `LLM4IA` recommends high-quality indexes for a new database instance without expensive retraining or fine-tuning.

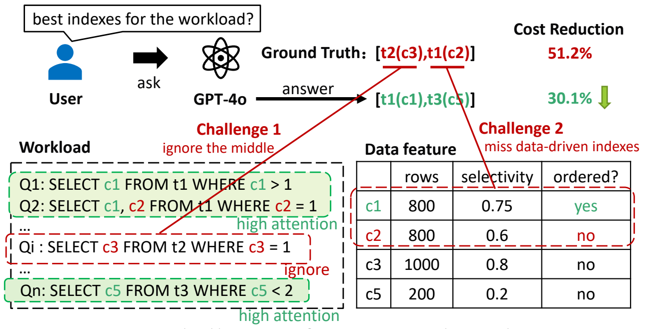
CCS Concepts

• **Information systems → Autonomous database administration**.

Keywords

Database; AI4DB; Index Advisor; LLM

ACM Reference Format:
Xian Lyu, Junbiao Zhang, Yihang Zheng, Guoliang Li, and Chen Lin. 2025. LLM4IA: Index Advising via Large Language Models. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3746252.3761475



```
Q1: SELECT c1 FROM t1 WHERE c1 > 1
Q2: SELECT c1, c2 FROM t1 WHERE c2 = 1
...
Qi : SELECT c3 FROM t2 WHERE c3 = 1
...
Qn: SELECT c5 FROM t3 WHERE c5 < 2
```

	rows	selectivity	ordered?
c1	800	0.75	yes
c2	800	0.6	no
c3	1000	0.8	no
c5	200	0.2	no

Figure 1: Challenges of LLMs in Index Selection

1 Introduction

Index selection is critical for optimizing database performance. Although many index advisors (IAs) have been proposed to free Database Administrators (DBAs) from the laborious index tuning process, existing IAs have limitations. Specifically, heuristic-based methods [4, 5] often yield suboptimal solutions because their fixed algorithm cannot capture the complex correlation between workload and data. Learning-based methods [1–3], while promising in adapting to different workloads, struggle to produce stable performance on new database instances due to the dependence on large amounts of high-quality training data.

The rise of large language models (LLMs) has brought new opportunities to solve complex problems in the database field. The ability of LLMs to combine extensive knowledge with step-by-step reasoning makes them powerful tools. Naturally, we ask if LLMs can be adopted as index advisors. However, as shown in Figure 1, even the most advanced commercial LLM, GPT-4o, faces two challenges.

Challenge 1: representing workloads in an LLM-friendly form. Workload representation is essential, enabling IAs to discover useful indices for a particular workload. Learning-based IAs adopt various workload representation strategies to transform a workload into a numerical embedding vector. Representing a workload is nontrivial for LLMs. On the one hand, LLMs are better at interpreting textual queries. The embedding vector derived by learning-based IAs is incomprehensible to LLMs. On the other hand, LLMs risk position bias and information loss when handling long contexts. Feeding the LLMs the original workload, which is a long list of queries and frequencies, prevents LLMs from capturing information truly related to index selection. As shown in Figure 1, GPT-4o selected indexes "`t1(c1)`" and "`t3(c5)`" because it placed higher attention on the beginning and ending parts of the workload, while ignoring relevant information in the middle that leads to the ground-truth index "`t2(c3)`".

*Chen Lin is the corresponding author.

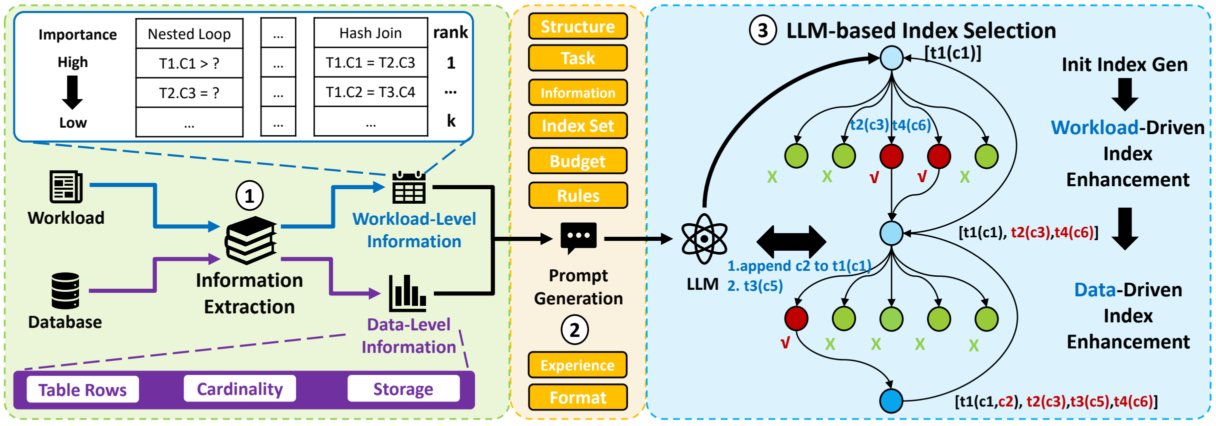


Figure 2: System Overview of LLM4IA

Challenge 2: finding the optimal index. Like DBAs, LLMs possess some database knowledge, enabling them to follow established guidelines to identify some adequate columns (e.g., frequently queried columns or highly selective columns) while also being prone to overlooking superior columns that can only be discovered through data-driven analysis. As shown in Figure 1, GPT-4o chose "c1" over "c2" in table "t1" because the index selectivity of "c1" is higher than that of "c2". The ground truth (optimal index) is "c2" since "c1" is ordered while "c2" is unordered. Unfortunately, it is difficult to determine whether a column is sorted from the database schema or the workload alone, and thus GPT-4o produced unsatisfying results.

To address these challenges, we propose LLM4IA, an LLM-based index advisor that can directly recommend indexes for any workload on any database instance.

To address Challenge 1, LLM4IA represents the workload in a concise textual description, avoiding numerical input entirely. LLM4IA first extracts essential predicates from query plans, eliminates the numerical values, and sorts them based on query frequency and estimated cost under each category. Therefore, key information relevant to index selection is combined and compacted to enhance LLMs' understanding of the impact of workload on indexes.

To address Challenge 2, LLM4IA adopts an iterative search method using LLMs. In each round, LLM4IA first makes multiple attempts to enhance the previous index set based on the workload representation or data characteristics, then summarizes all effective candidates to obtain a better index set for the next round. Therefore, the index set is self-evolved, uncovering potential good indexes.

Furthermore, LLM4IA utilizes a set of strategies to reduce the overhead of LLMs, including the token budget and the inference time. LLM4IA shortens the predicate list by truncation and abstracts table and column names to minimize the number of tokens in the input. LLM4IA also employs parallel enhancement in the iterative search to accelerate LLM inference.

Contributions. Compared with heuristic-based methods, LLM4IA is less likely to fall into suboptimal solutions. LLM4IA outperforms the near-optimal IA Extend by 5%-10% on TPC-H and TPC-DS benchmarks. Compared with learning-based methods, LLM4IA can provide index recommendations on new database instances, without specific training on the database instance. Compared with directly using GPT4-o with the original workload, LLM4IA reduces the workload execution time by 20%-30% on TPC-H and 70%-80% on TPC-DS, while reducing the token overhead by 60% - 70%. The demonstration video and the codes are available at https://github.com/XMUDM/LLM4IA.

2 Methodology

As shown in Figure 2, LLM4IA consists of three modules.

2.1 ① Information Extraction

This module extracts index-relevant information from workload-level and data-level to retain effective contextual information while reducing the input size of LLMs.

Workload-level information. **Step 1: Predicate extraction**. Proper alignment of indexes with frequently executed predicates can significantly enhance query performance. Therefore, our core idea is to utilize predicates to obtain a brief and informative workload representation rather than the lengthy and obscure original workload. Since the query plans contain more detailed information, LLM4IA first transforms each query in the workload into a query plan. The predicates are the filtering conditions of each node in the query plan. Then, LLM4IA classifies the predicates according to the types of their corresponding node. For example, predicates "T1.C1>1" and "T2.C3=1" are grouped under node type "Nested Loop". **Step 2: Non-numerical transformation**. Since there are infinitely many potential values for the literals, LLMs may misinterpret the scales or ranges because they are not extensively trained on rare values. To generalize the literals, LLM4IA replaces them with the placeholder "?". **Step 3: Predicate ranking**. LLM4IA calculates "estimated cost × query frequency" to order all predicates in the same type, where "estimated cost" refers to the estimated cost of the node in the query plan, and the "query frequency" refers to the frequency of the corresponding query. **Step 4: Predicate filtering**. A lengthy input may raise difficulties for LLMs to grasp crucial information and increase the cost of calling LLM APIs. Since the predicates with lower frequency or smaller estimated costs provide less significant performance benefits, we reserve the top k predicates in the ranking.

Data-level information. To enable the LLM to perceive index-related database statistics, LLM4IA extracts the number of recodes in each table, the cardinality of each column, and the storage occupancy of single-column indexes from the database, and formalizes them into textual descriptions.

Information Compressing. To reduce the token budget, LLM4IA removes tables and columns not targeted in the current workload. LLM4IA also abstract the table names and column names to simplify the text representation. For example, a table named "lineitem" and a column named "l_extendedprice" in the TPC-H benchmark are abbreviated as "t_i" and "C_i", respectively. The above compression techniques are applied to both workload-level and data-level information.

2.2 ② Prompt Generation

This module generates the prompt feeding LLMs, which comprises eight parts. **1 Database Structure Information**. This part describes the database structure, e.g., which table a certain column belongs to. **2 Task Description**. We generate different task descriptions according to different index recommendation stages, e.g., initial index set generation, workload-driven index enhancement, and data-driven index enhancement mentioned in Section 2.3. **3 Key Information for Index Recommendation**. This part includes the workload-level and data-level information extracted in Section 2.1. The exact input in this part differs for different index recommendation stages. More details are provided in Section 2.3. **4 Current Index Set**. This part inputs the initial index set or the index set obtained from the previous step. The initial index set is empty. **5 Storage Budget**. This part includes the storage budget predefined by the user. **6 Index Recommendation Rules**. This part contains some predefined rules for index recommendation (e.g. keep the storage cost of the index within budget). Users can also customize their own rules. **7 Index Recommendation Experience**. This part contains all indexes that LLM4IA found during the index selection process up to the current step that are not profitable. We ask the LLM not to recommend indexes in the experience in the next step, avoiding repeated inefficient searches. **8 Output Format Requirements**. This part standardizes the output of the LLM to facilitate subsequent processing.

2.3 ③ LLM-based Index Selection

This module calls LLMs to perform index selection.

Step 1: Initial Index Set Generation By default, LLM generates the initial index set. The prompt is generated as in Section 2.2, where the "Key Information for Index Recommendation" part contains workload and data-level information and the "Task Description" part asks the LLM to recommend the index configuration that it believes will yield the most benefit within a given storage budget based on the input information. LLM4IA also allows users to generate the initial index set by other existing index advisors.

Step 2: Two-stage Iterative Index Enhancement It is difficult to generate high-quality indexes by making a single call to the LLM. Therefore, we propose an iterative enhancement process based on the LLM to optimize the index set until it cannot be improved. Furthermore, the input of the LLM contains both workload information and data-related information. However, the workload information is much longer than the data-related information in length, making it difficult for the LLM to focus on the data-related parts. Therefore, our iterative enhancement process is divided into two stages, one stage focusing on workload information and the other stage focusing on data-related information.

Step 2.1: Workload-driven index enhancement. The enhancement of this stage focuses on finding good indexes from the workload. Therefore, the "key information for index recommendation" part in the prompt only includes the workload-level information. Based on the initial index set, LLM4IA seeks a better index set within the given storage budget by either "adding a new index", "expanding an existing index with additional columns", or "replacing an index with an unseen index", repeatedly (Described in "Task Description"). During each round of index enhancement, we call LLM multiple times to generate diverse candidates. We verify the effectiveness of each candidate by calling the what-if optimizer. As shown in Figure 2, the red node means that the node has found a better index configuration, while the green node means it has not. The user pre-defines the maximal number of attempts. To reduce the time overhead caused by multiple LLM calls, LLM4IA performs multiple explorations in a single index enhancement step in parallel.

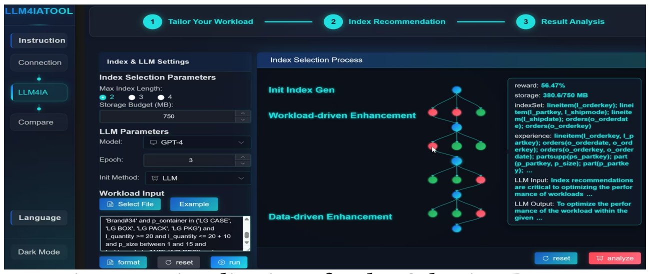
Due to the inevitable hallucination issue of LLMs during the generation process, to ensure the correctness of the final recommended indexes and reduce unnecessary enhancement iterations, LLM4IA filters out "invalid index" (e.g. indexes that do not provide benefits, indexes where the columns do not belong to the table of the index, and multi-column indexes with the number of columns exceeding the limit). Furthermore, to fully utilize the multiple attempts from each round, we develop a memory mechanism, i.e., collect and record the indexes explored in each round that do not provide benefits. In the subsequent exploration, this "experience" part in the prompt will significantly prevent the LLM from repeatedly exploring useless indexes, thereby making the process more efficient and effective.

Next, instead of selecting the optimal candidate, LLM4IA combines all valid candidates by sorting the indexes according to their benefits and merging the indexes from high benefits to low benefits until the storage budget is met. An enhanced index set is obtained after the combination.

Step 2.2 Data-driven index enhancement. Step 2.1 alone is insufficient because the estimated query plans may not be accurate. There are some indexes that, although rarely appearing in the workload, are good indexes due to their excellent data-level characteristics. Therefore, it is impossible to find all possible good indexes based only on workload-driven index enhancement, and we need to increase the recall rate, i.e., discover all potentially good indexes, to determine the optimal indexes. To avoid being misled by workload-level information, the "key information for index recommendation" of this stage only contains data-level information. Other details of this stage are similar to those in **Step 2.1**.

3 Demonstration

In this section, we first demonstrate the index iteration enhancement process of LLM4IA. Then, to verify that LLM4IA surpasses existing IAs, we demonstrate a detailed comparative evaluation on index qualities. Users can test LLM4IA on several typical analytical benchmark tests (e.g., TPC-H and TPC-DS) using their customized workloads.



Index & LLM Settings

Index Selection Parameters

Max Index Length: ● 2 ○ 3 ○ 4

Storage Budget (MB): 750

LLM Parameters

Model: GPT-4

Epoch: 3

Init Method: LLM

Workload Input

'Brand#34' and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG') and l_quantity >= 20 and l_quantity <= 20 + 10 and p_size between 1 and 15 and

Index Selection Process

reward: 56.47%

storage: 380.6/750 MB

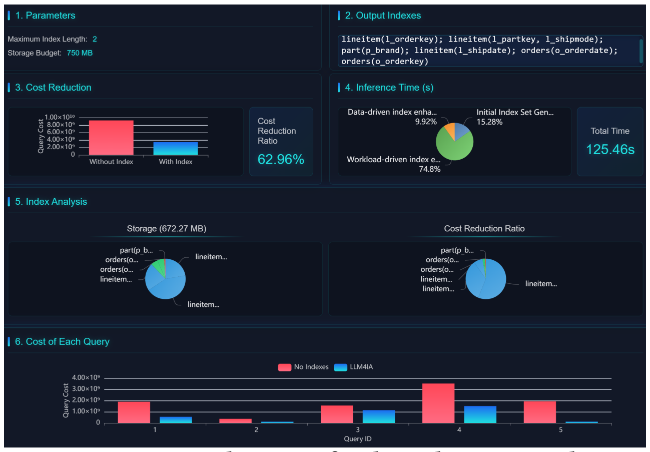
indexSet: lineitem(l_orderkey); lineitem(l_partkey, l_shipmode); lineitem(l_shipdate); orders(o_orderdate); orders(o_orderkey)

experience: lineitem(l_orderkey, l_partkey); orders(o_orderdate, o_orderkey); orders(o_orderkey, o_orderdate); partsupp(ps_partkey); part(p_partkey, p_size); part(p_partkey); ...

LLM Input: Index recommendations are critical to optimizing the performance of workloads ...

LLM Output: To optimize the performance of the workload within the given ...

Figure 3: Visualization of Index Selection Process



1. Parameters

Maximum Index Length: 2

Storage Budget: 750 MB

2. Output Indexes

```
lineitem(l_orderkey); lineitem(l_partkey, l_shipmode);
part(p_brand); lineitem(l_shipdate); orders(o_orderdate);
orders(o_orderkey)
```

Cost Reduction Ratio 62.96%

Total Time 125.46s

Figure 4: Visualization of Index Selection Result

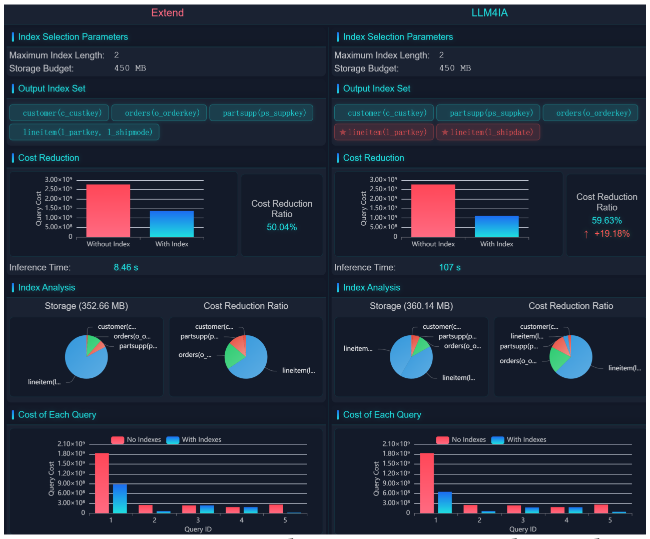
3.1 Visualization of Index Selection of LLM4IA

The visualization of this part includes two aspects. **1 Visualization of the index selection process**. As shown in Figure 3, after users customize the parameters for index selection (e.g., max index length, storage budget), LLM4IA parameters (e.g. LLM model, the number of attempts in each enhancement), and upload the workload, an index selection flowchart is generated. This flowchart illustrates how LLM4IA iteratively enhances the initial index set step by step to obtain the final recommended index. Users can hover their mouse over specific nodes to view details (e.g., the current index set, storage consumption, and cost reduction ratio).

2 Visualization of the index selection results. As shown in Figure 4, users can get a result analysis interface for a comprehensive visualization of the results, including (1) the parameter configuration, e.g., maximal index length and storage budget. (2) the output indexes by LLM4IA, (3) the cost reduction ratio w.r.t. the null index, which is the execution time ratio of running the workload given the selected indexes v.s. given no indexes. (4) The inference time, including the end-to-end overhead and the time overhead of each stage. (5) Index analysis, including the storage consumption and cost reduction ratio of each index. (6) Cost of each query, which is the execution time before (i.e., null index) and after LLM4IA for each query in the workload.

3.2 Comparative Evaluation

This part allows users to select an existing IA to compare its recommendation effectiveness with that of LLM4IA. Similar to 3, users need to customize the index selection parameters, LLM4IA parameters, and the test workload. Then, users can get a detailed comparison report. As shown in Figure 5, we compare the recommended indexes of two IAs from multiple aspects as Section 3.1. The differences in the output indexes are highlighted in red.



Extend	LLM4IA
Index Selection Parameters	Index Selection Parameters
Maximum Index Length: 2	Maximum Index Length: 2
Storage Budget: 450 MB	Storage Budget: 450 MB
Output Index Set	Output Index Set
customer(c_custkey) orders(o_orderkey) partsupp(ps_suppkey) lineitem(l_partkey, l_shipmode)	customer(c_custkey) partsupp(ps_suppkey) orders(o_orderkey) ★lineitem(l_partkey) ★lineitem(l_shipdate)
Cost Reduction Ratio 50.04%	Cost Reduction Ratio 59.63% ↑ +19.18%
Inference Time: 8.46 s	Inference Time: 107 s

Figure 5: Comparison between LLM4IA and Extend

To further investigate the index recommendation performance of LLM4IA, in TPC-H 1GB, we take Extend [4], which is considered near-optimal in the literature, for comparison in Figure 5 as an example. Under the given storage budget, Extend selects index "lineitem(l_partkey, l_shipmode)" with a higher benefit-to-storage ratio, but the high storage consumption of index "lineitem(-l_partkey, l_shipmode)" prevents the selection of index "linei-tem(l_shipdate)", thus falling into a suboptimal solution. On the contrary, LLM4IA selects index "lineitem(l_partkey)" with a lower benefit-to-storage ratio, but the lower storage consumption of index "lineitem(l_partkey)" allows for the selection of index "lineitem(l_shipdate)". Finally, because the combined benefits of index "lineitem(l_shipdate)" and index "lineitem(l_partk-ey)" exceed that of index "lineitem(l_partkey, l_shipmode)", LLM4IA ultimately achieves a higher performance.

We have conducted extensive experiments in the open-source database testing benchmarks TPC-H and TPC-DS and found that LLM4IA stably outperforms the near-optimal IA Extend by 5%-10%. Learning-based IAs have two limitations: (1) they cannot be used on a new db instance without hours of training, (2) their performance degrades severely under workload drift. Therefore, we use the same testing workload as the training workload, and LLM4IA exceeds the SOTA learning-based IA SWIRL [1] by 11%.

4 Conclusion

LLM4IA presents a novel paradigm of employing LLMs for index selection by feeding LLMs with a concise textual description of the key predicates and iteratively enhancing and summarizing indexes for self evolution. LLM4IA applies directly to any database instance and workload, which is promising. One limitation is that the inference time of LLM4IA is still large, which leaves as a future direction for improvement.

5 Acknowledgments

Chen Lin is the corresponding author and is supported by the Natural Science Foundation of China (No.62372390,62432011).

References

[1] Jan Kossmann, Alexander Kastius, and Rainer Schlosser. 2022. SWIRL: Selection of Workload-aware Indexes using Reinforcement Learning.. In *EDBT*, Vol. 2. 155–2.

[2] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2020. An index advisor using deep reinforcement learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2105–2108.

[3] Zahra Sadri, Le Gruenwald, and Eleazar Lead. 2020. DRLindex: deep reinforcement learning index advisor for a cluster database. In *Proceedings of the 24th Symposium on International Database Engineering & Applications*. 1–8.

[4] Rainer Schlosser, Jan Kossmann, and Martin Boissier. 2019. Efficient scalable multi-attribute index selection using recursive strategies. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1238–1249.

[5] Gary Valentin, Michael Zuliani, Daniel C Zilio, Guy Lohman, and Alan Skelley. 2000. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*. IEEE, 101–110.