Demonstration of ViTA: Visualizing, Testing and Analyzing Index Advisors

Wei Zhou
Chen Lin*
School of Informatics, Xiamen University
China
weizhou@stu.xmu.edu.cn
chenlin@xmu.edu.cn

Xuanhe Zhou
Guoliang Li
Department of Computer Science and Technology, Tsinghua University
China
zhouxuan19@mails.tsinghua.edu.cn
liguoliang@tsinghua.edu.cn

Tianqing Wang
Huawei Company
China
wangtianqing2@huawei.com

ABSTRACT

Index advisors have become an essential tool to optimize index selection and accelerate query processing. Various index advisors have been developed in recent years, and comprehensively assessing their performance from multiple aspects is necessary. In this demonstration, we introduce **ViTA**, a user-friendly and informative tool for interactively **Vi**sualizing, **T**esting, and **A**nalyzing index advisors. For a user-given workload, **ViTA** can visualize the main steps of the index selection procedure in ten existing index advisors to facilitate the management of index advisors. Moreover, **ViTA** can assess the index advisor's robustness w.r.t. workload drift by generating testing workloads, i.e., potentially future workloads that may damage the index advisor's performance. Finally, **ViTA** provides a comparative analysis across index advisors on four aspects, including the index advisor's utility (i.e., the ratio of the reduced workload cost), robustness (i.e., the performance under dynamic workload), overhead (i.e., the time to acquire the final configuration), and scalability (i.e., the volume of the enumerated index candidates). Therefore, **ViTA** can thoroughly compare existing index advisors to help users determine the most suitable index advisor that meets their requirements. **ViTA** is now being integrated into the openGauss platform as a plug-in[1].

CCS CONCEPTS

• **Information systems** → **Database utilities and tools**.

KEYWORDS

Index Tuning; AI4DB; Database Performance Assessment

*Chen Lin is the corresponding author. The project was sponsored by CCF-Huawei Populus Grove Fund (CCF-HuaweiDB2022002), and the Natural Science Foundation of China (No. 61972328).

[1]https://gitee.com/opengauss/openGauss-DBMind

ACM Reference Format:
Wei Zhou, Chen Lin, Xuanhe Zhou, Guoliang Li, and Tianqing Wang. 2023. Demonstration of ViTA: Visualizing, Testing and Analyzing Index Advisors. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3583780.3614738

1 INTRODUCTION

Indexing is crucial to optimize workload performance. Traditionally, expert database administrators analyze the workload characteristics and manually create indexes. To automate this labor-intensive process, index advisors, which automatically select a set of indexes for a given workload, have been extensively studied in both academia and industry [2–4, 7–9, 11, 12, 14–17].

Conventionally, index advisors are evaluated using synthetic workloads, and their performance is measured by the reduced workload cost achieved under the recommended indexes [7]. Nevertheless, this evaluation scheme is neither effective nor comprehensive, as users require interactive analysis to understand the index advisor's performance across multiple aspects. Specifically, conventional evaluation methods are limited in the following ways.

- *Little Knowledge of the Index Advisor's Selection Procedure.* In a quantitative evaluation, all index advisors are treated as opaque boxes, and only the final output, i.e., index configuration, is presented. It might be difficult even for expert DBAs to analyze the reasons when sub-optimal index configurations are presented.
- *Limited Diversity of Assessment Workloads.* In the literature, most evaluations are conducted on workloads generated from a set of fixed templates. The index advisor's robustness is not well captured, i.e., whether the index advisor will maintain high performance on a future workload slightly different from the original ones due to workload drift [10].
- *Lack of Multi-Aspect Analysis.* Previous evaluations have focused primarily on the reduced workload cost while overlooking other desirable aspects [5], such as the robustness of index advisors under dynamic workloads, the overhead involved in selecting the final index configurations, and the scalability in handling workloads with many index candidates.

In this demonstration, we propose **ViTA**, a system towards **Vi**sualizing, **T**esting, and **A**nalyzing index advisors. To address the aforementioned limitations, **ViTA** provides three unique functions. (1) For any user-given workload, **ViTA** visualizes the main steps of the index selection procedure in ten existing index advisors [4, 7–9, 11, 12, 14, 17]. Thus, **ViTA** is a user-friendly tool to facilitate the management of index advisors. (2) **ViTA** goes beyond evaluating indexing performance on static workload to assess the index advisor's robustness w.r.t. workload drifts. For any user-input workload, **ViTA** integrates a *learned component* to automatically generate testing workloads, which are (a) potential future workloads according to user-defined perturbation types, i.e., common patterns of daily query changes and (b) aim to degrade the index advisor's performance with *limited* knowledge, i.e., no explicitly knowing the inherent characteristics of the index advisors. The index performance on the input workload and testing workloads are compared to measure the index advisor's robustness. (3) **ViTA** provides comprehensive analysis across index advisors on multiple aspects, such as the index advisor's utility (i.e., the ratio of the reduced workload cost), robustness (i.e., the performance over dynamic workload), overhead (i.e., the time to acquire the final configuration) and scalability (i.e., the volume of the enumerated index candidates). Therefore, **ViTA** can thoroughly compare existing index advisors to help users determine which index advisor best meets their requirements.

2 SYSTEM OVERVIEW

As shown in Figure 1, **ViTA** consists of three modules.

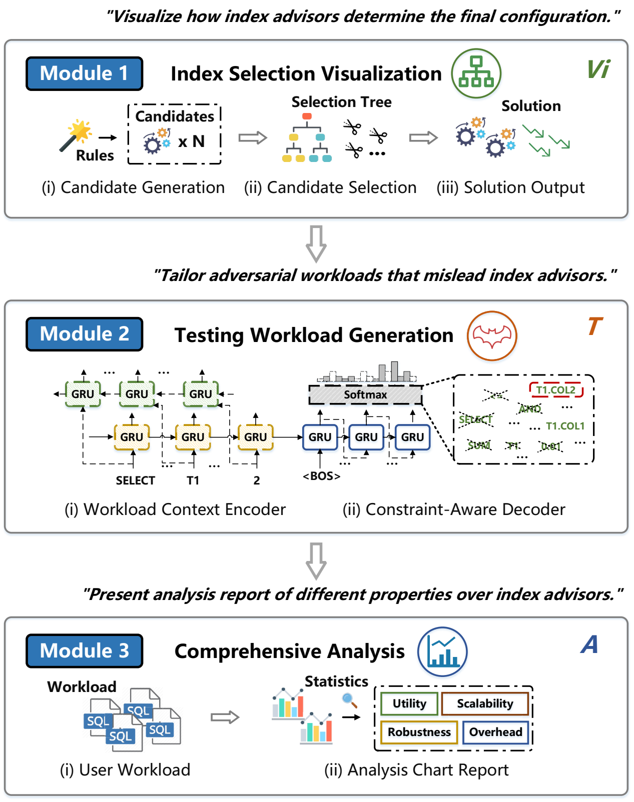

"Visualize how index advisors determine the final configuration."
Module 1
Index Selection Visualization
Vi
Candidates
x N
Rules
Selection Tree
Solution
(i) Candidate Generation
(ii) Candidate Selection
(iii) Solution Output
"Tailor adversarial workloads that mislead index advisors."
Module 2
Testing Workload Generation
T
GRU
Softmax
T1.COL2
T1.COL1
SELECT
T1
2
<BOS>
(i) Workload Context Encoder
(ii) Constraint-Aware Decoder
"Present analysis report of different properties over index advisors."
Module 3
Comprehensive Analysis
A
Workload
SQL
Statistics
Utility
Scalability
Robustness
Overhead
(i) User Workload
(ii) Analysis Chart Report

Figure 1: System Overview of ViTA

Module 1: Index Selection Visualization. In this module, **ViTA** asks the user to input a workload and specify an index advisor. Then, **ViTA** visualizes the general selection procedure in the index advisors. (1) *Candidate Generation.* **ViTA** presents the initial index candidates generated by the index advisor, which are promising indexes considering the syntactic information. (2) *Candidate Selection.* **ViTA** formulates the selection procedure as a hierarchical tree, where each layer corresponds to a selection step, and nodes in the layer represent the index candidates in this step. Note that for index advisors based on reinforcement learning, we formulate the selection procedure in the last episode. (3) *Solution Output.* **ViTA** displays the final index configuration and measures the index advisor's performance by the cost reduction ratio of the workload under the recommended indexes.

Module 2: Testing Workload Generation. In this module, **ViTA** accepts a workload and assesses the robustness of index advisors under workload drift. **ViTA** integrates a *learned component* to automatically generate testing workloads to mimic representative workload drifts. Then, **ViTA** compares the index advisor's performances on the original input workload and the generated testing workload. More technical details will be presented in Section 3.

Module 3: Comprehensive Analysis. In this module, **ViTA** provides comprehensive analysis across index advisors. Four aspects are considered and visualized, including *indexing utility, indexing robustness, indexing overhead,* and *indexing scalability.* Specifically, to reflect the indexing utility, **ViTA** computes the cost reduction ratio under the recommended indexes. To reflect indexing robustness, **ViTA** compares the cost reduction ratio on the original input workload and generated testing workload. To reflect indexing overhead, **ViTA** records the time budget on the overall index selection procedure. To reflect indexing scalability (i.e., the effectiveness of an index advisor on a large workload), **ViTA** counts the number of index candidates enumerated during the index selection procedure.

3 TESTING WORKLOAD GENERATION

Given the user's input workload, **ViTA** generates the testing workloads with two properties. (1) *Similarity*: Firstly, the testing workloads are derived from the input workload by adding small perturbations based on a user-defined perturbation type. Workload drifts are observed in real-world production systems and open-source benchmarks, where many queries are variants of a small number of templates. For example, 1.7 billion queries are generated from 31 million templates based on an analysis over a real-world production system [1]. To represent typical workload drifts, **ViTA** defines three perturbation types (Section 3.1) and asks the user to choose the *perturbation constraint*, i.e., the possible perturbation type and the degree of perturbation that are likely to occur in future workloads. (2) *Adversary*: Secondly, the testing workload is generated to steer the index advisor to choose a sub-optimal solution and cause a performance regression. The motivation is: guiding the generation with an adversarial aim [6] can be more effective in revealing bad cases and helping the user analyze index advisors. Towards this end, **ViTA** proposes an encoder-decoder model (Section 3.2) to modify queries in the input workload, which is trained in a reinforcement learning framework with a novel reward function to degrade the performance of index advisors. Furthermore, when **ViTA** invokes the generation model, a *Constraint-Aware Reference Tree* is introduced to ensure the output is syntactically correct and conforms to the perturbation type. Thus, the generated testing workloads can effectively evaluate an index advisor's robustness under workload drifts.

3.1 Perturbation Types

We summarize three perturbation types according to the observations from daily workload drifts. (1) *Value Only Perturbation* reflects the most common template-based workload drift when queries are variants of the same template with different values. For example, an online retailer may issue a series of queries to compare the sales figures of the same product in different seasons, pricing strategies, or marketing campaigns. (2) *Column Consistent Perturbation* mimics the workload change when users operate on the same set of columns. For instance, a customer shopping on an e-commerce website may change the order of columns in the search results to display products according to their preferences and requirements. (3) *Shared Table Perturbation* simulates the scenario when users change the payload (i.e., columns in the SELECT clause) or add new predicates to the original queries. For example, a sales analyst may perform exploratory analysis with different filter predicates for the date range, product category, or customer demographics to uncover trends or patterns and assist inventory management decisions.

Since a SQL statement is composed of tokens, where tokens can be roughly categorized into seven token types, i.e., `Column, Table, Value, Keyword, Conjunction, Operator, Aggregator`, the three perturbation types can be defined at the token level, and the legitimate perturbation token types for each perturbation type are shown in Table 1.

Table 1: Legitimate token types w.r.t. perturbation types

Perturbation	Column	Value	Conjunction	Operator	Aggregator
Shared Table	✓	✓	✓	✓	✓
Column Consistent	✓	✓	-	-	-
Value Only	-	✓	-	-	-

3.2 The Generation Model

The generation model is an encoder-decoder framework [13]. The encoder reads the original queries in the input workload, and the decoder generates the perturbed queries based on the encoder's output and the previously generated tokens. As shown in Figure 2, a novel *Constraint-Aware Reference Tree* is introduced in the generation process to preserve the perturbation constraints and syntactic correctness of the output.

Initializing Constraint-Aware Reference Tree. **`ViTA`** parses the queries in the input workload and initializes a CAR-tree (Constraint-Aware Reference Tree) according to a set of predefined Backus–Naur Form grammar rules (BNF rules). The root node of the CAR-tree represents the SQL statement. Each non-leaf node corresponds to a non-terminal symbol in a BNF rule (i.e., a clause or a token type). And each leaf node corresponds to an actual token in the query.

As displayed in Figure 2, the input query is `SELECT T1.a FROM T1 WHERE T1.b > 0.02`, according to one of the BNF rules defined in **`ViTA`** `SQL ::= SELECT FROM WHERE [GROUPBY] [HAVING] [ORDERBY]`, the root node will be `SQL`, the three nodes in the first layer will be `SELECT, FROM, WHERE`. According to another BNF rule `SELECT ::= "select" (column ("," column)? | SQL)`, the children of `SELECT` will be "`select`", `column` and the child node of `column` will be `T1.a`, which is a leaf node.

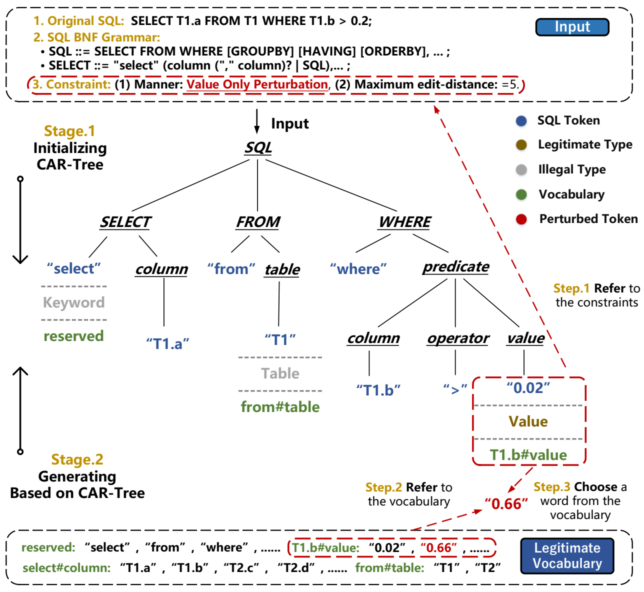

1. Original SQL: SELECT T1.a FROM T1 WHERE T1.b > 0.2;
2. SQL BNF Grammar:
• SQL ::= SELECT FROM WHERE [GROUPBY] [HAVING] [ORDERBY], ... ;
• SELECT ::= "select" (column ("," column)? | SQL),... ;
3. Constraint: (1) Manner: Value Only Perturbation, (2) Maximum edit-distance: =5.
Input
Input
Stage.1 Initializing CAR-Tree
SQL
SELECT
FROM
WHERE
"select"
Keyword
reserved
column
"T1.a"
"from"
table
"T1"
Table
from#table
"where"
predicate
column
"T1.b"
operator
">"
value
"0.02"
Value
T1.b#value
● SQL Token
● Legitimate Type
● Illegal Type
● Vocabulary
● Perturbed Token
Step.1 Refer to the constraints
Stage.2 Generating Based on CAR-Tree
Step.2 Refer to the vocabulary
"0.66"
Step.3 Choose a word from the vocabulary
reserved: "select" , "from" , "where" , ...
T1.b#value: "0.02" , "0.66" , ...
select#column: "T1.a" , "T1.b" , "T2.c" , "T2.d" , ...
from#table: "T1" , "T2"
Legitimate Vocabulary

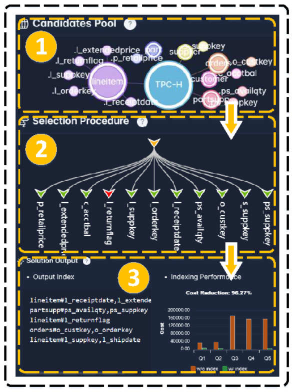
Figure 2: Example of Constraint-Aware Reference Tree

After initializing the CAR-tree, a *legitimate vocabulary* is initialized to include each non-leaf node's legitimate tokens. For example, the legitimate tokens for `from#table` include the tables in the current database and the legitimate tokens for `T1.b#value` include a set of sampled values from `T1.b`. The CAR-tree and its associated legitimate vocabulary will be dynamically updated during generation.

Generating based on Constraint-Aware Reference Tree. The decoder generates a token at a step and performs the following operations in each step. Firstly, given the input query and previously generated tokens, the decoder decides whether the current position of the input query can be modified according to the perturbation constraint. If the perturbation constraint forbids modification, the decoder will directly copy the token from the input query's current position and proceed to the next position. Otherwise, the decoder will traverse to the lowest ancestor of the current leaf node on the CAR-tree based on the perturbation constraint. Then, the decoder will fetch the legitimate vocabulary associated with the current node and draw a token from the legitimate vocabulary[2]. Note that the current token itself is also included in the vocabulary if legitimate. After generating the current token, the CAR-tree, and the corresponding legitimate vocabulary will be updated, e.g., the leaf node is replaced with the changed token.

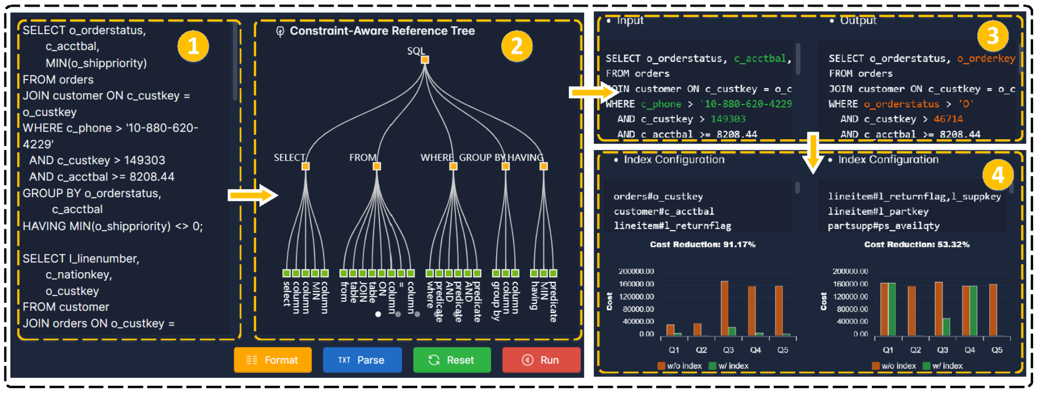
As shown in Figure 2, the input query is `SELECT T1.a FROM T1 WHERE T1.b > 0.02`, previously generated tokens are `SELECT T1.a FROM T1 WHERE T1.b >`, the perturbation constraint is `Value-Only Perturbation` which means that only tokens with the type `Value` in the input query can be altered. In this case, the token from the

[2]Due to page limit, we omit details of the generation probability of drawing a token from a vocabulary, which is standard in the encoder-decoder framework.



```
lineitem#l_receiptdate,l_extende
partsupp#ps_availqty,ps_suppkey
lineitem#l_returnflag
orders#o_custkey,o_orderkey
lineitem#l_suppkey,l_shipdate
```

(a) Index Selection Visualization



```
SELECT o_orderstatus,
       c_acctbal,
       MIN(o_shippriority)
FROM orders
JOIN customer ON c_custkey =
o_custkey
WHERE c_phone > '10-880-620-
4229'
  AND c_custkey > 149303
  AND c_acctbal >= 8208.44
GROUP BY o_orderstatus,
         c_acctbal
HAVING MIN(o_shippriority) <> 0;

SELECT l_linenumber,
       c_nationkey,
       o_custkey
FROM customer
JOIN orders ON o_custkey =
```

```
SELECT o_orderstatus, c_acctbal,
FROM orders
JOIN customer ON c_custkey = o_c
WHERE c_phone > '10-880-620-4229
  AND c_custkey > 149303
  AND c_acctbal >= 8208.44
```

```
SELECT o_orderstatus, o_orderkey
FROM orders
JOIN customer ON c_custkey = o_c
WHERE o_orderstatus > 'O'
  AND c_custkey > 46714
  AND c_acctbal >= 8208.44
```

```
orders#o_custkey
customer#c_acctbal
lineitem#l_returnflag
```

```
lineitem#l_returnflag,l_suppkey
lineitem#l_partkey
partsupp#ps_availqty
```

(b) Testing Workload Generation

Figure 3: The Index Selection Visualization and Testing Workload Generation Interface

current position should be drawn from the legitimate vocabulary of `T1.b#value`.

Training the Generation Model. Reinforcement learning is adopted to train the generation model. The decoder acts as an *agent* that generates a testing workload in each episode based on the *state*, i.e., the encoder's output given the input workload and receives a *reward*. To guide the generated queries to be an adversary of the index advisor, we formulate the reward function, which calculates the performance drop of the generated workload compared with the input,

$$r = 1 - \frac{u(\mathcal{W}', \mathbf{d}, f)}{u(\mathcal{W}, \mathbf{d}, f)}, \tag{1}$$

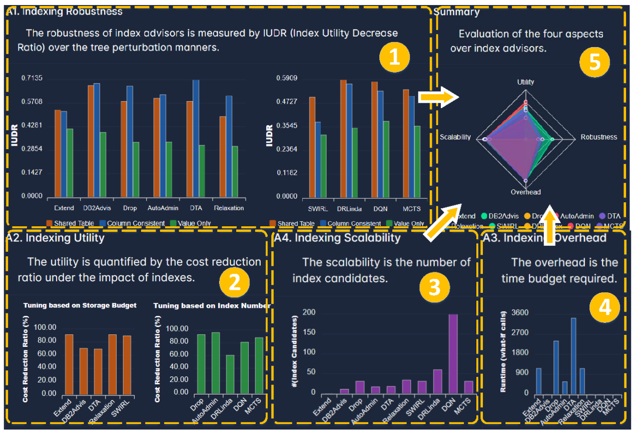
where $\mathcal{W}$ denotes the input workload, $\mathcal{W}'$ denotes the generated testing workload, $\mathbf{d}$ denotes the database, f denotes the index advisor and $f(\mathcal{W}, \mathbf{d})$ is the set of index advisor's recommended indexes, the cost reduction ratio $u(\mathcal{W}, \mathbf{d}, f) = 1 - \frac{c\left(\mathcal{W}, \mathbf{d}, f(\mathcal{W}, \mathbf{d})\right)}{c\left(\mathcal{W}, \mathbf{d}, \emptyset\right)}$, $\emptyset$ denotes null index, $c(\mathcal{W}, \mathbf{d}, f(\mathcal{W}, \mathbf{d}))$ is the cost of running $\mathcal{W}$ on $\mathbf{d}$ with f's recommended indexes. Thus, if the index advisor performs better on the original workload $\mathcal{W}$ than on the testing workload $\mathcal{W}'$, i.e., the workload cost reduction ratio $u(\mathcal{W}, \mathbf{d}, f)$ is larger than $u(\mathcal{W}', \mathbf{d}, f)$, then $r > 0$.

4 DEMONSTRATION

The demonstration video is available at https://youtu.be/RfV4ylOxpcc. In the demonstration, the user will use the web interface to assess the performance of ten index advisors on the typical workloads, e.g., TPC-H, TPC-DS, and JOB. The demonstration comprises three steps, as shown in Figure 3 and Figure 4.

(1) **Index Selection Visualization**. Figure 3 (a) shows the result page, which displays the *Initial Candidates Pool* (❶) in a forced directed tree. The root node denotes the benchmark TPC-H, the children nodes represent each table in the database, and the leaf nodes correspond to the initial index candidates. The *Selection Procedure* (❷) is shown in a hierarchical tree with each layer as a selection step. All the nodes denote the index candidates at this step, and the selected one corresponds to the red node. More detailed information, e.g., the value of the workload cost, is presented when the mouse hovered. The *Solution Output* (❸) is presented, and the cost reduction ratio of queries over the workload is reported in a bar plot.

(2) **Testing Workload Generation**. As shown in Figure 3 (b), after the user specifies an input workload (❶), an index advisor and a perturbation constraint (i.e., the perturbation type and degree of perturbation), the user can click "Parse" to show the CAR Tree (❷) of each query introduced in Section 3. Then they can click "Run" to view the result, check the difference between the input and testing workload (tokens highlighted in green and red in ❸), and analyze the comparative workload cost on input and testing workloads with and without the recommended indexes in a bar plot (❹).



A1. Indexing Robustness

The robustness of index advisors is measured by IUDR (Index Utility Decrease Ratio) over the tree perturbation manners.

Summary

Evaluation of the four aspects over index advisors.

A2. Indexing Utility

The utility is quantified by the cost reduction ratio under the impact of indexes.

A4. Indexing Scalability

The scalability is the number of index candidates.

A3. Indexing Overhead

The overhead is the time budget required.

Figure 4: The Comprehensive Analysis Interface

(3) **Comprehensive Analysis**. Finally, Figure 4 shows a comprehensive comparative analysis among different index advisors on four aspects, i.e., indexing robustness (❶), indexing utility (❷), indexing scalability (❸) and indexing overhead (❹) in the bar plots. The overall performances are summarized in a radar chart (❺).

REFERENCES

[1] Amirhossein Aleyasen, Mark Morcos, Lyublena Antova, Marc Sugiyama, Dmitri Korablev, Jozsef Patvarczki, Rima Mutreja, Michael Duller, Florian M. Waas, and Marianne Winslett. 2022. Intelligent Automated Workload Analysis for Database Replatforming. In *SIGMOD Conference*. ACM, 2273–2285.

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

[3] S. Chaudhuri and V. Narasayya. 2020. Anytime Algorithm of Database Tuning Advisor for Microsoft SQL Server. https://www.microsoft.com/en-us/research/publication/anytime-algorithm-of-database-tuning-advisor-for-microsoft-sql-server/.

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

[5] Sudipto Das, Miroslav Grbic, Igor Ilic, Isidora Jovandic, Andrija Jovanovic, Vivek R. Narasayya, Miodrag Radulovic, Maja Stikic, Gaoxiang Xu, and Surajit Chaudhuri. 2019. Automatically Indexing Millions of Databases in Microsoft Azure SQL Database. In *SIGMOD Conference*. ACM, 666–679.

[6] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR (Poster)*.

[7] 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.

[8] Jan Kossmann, Alexander Kastius, and Rainer Schlosser. 2022. SWIRL: Selection of Workload-aware Indexes using Reinforcement Learning. In *EDBT*. OpenProceedings.org, 2:155–2:168.

[9] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2020. An Index Advisor Using Deep Reinforcement Learning. In *CIKM*. ACM, 2105–2108.

[10] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts. In *SIGMOD Conference*. ACM, 1920–1933.

[11] Zahra Sadri, Le Gruenwald, and Eleazar Leal. 2020. DRLindex: deep reinforcement learning index advisor for a cluster database. In *IDEAS*. ACM, 11:1–11:8.

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

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *NIPS*. 3104–3112.

[14] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. 2000. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In *ICDE*. IEEE Computer Society, 101–110.

[15] Kyu-Young Whang. 1985. Index Selection in Relational Databases. In *FODO*. Plemum Press, New York, 487–500.

[16] 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 Conference*. ACM, 1528–1541.

[17] Xuanhe Zhou, Luyang Liu, Wenbo Li, Lianyuan Jin, Shifu Li, Tianqing Wang, and Jianhua Feng. 2022. AutoIndex: An Incremental Index Management System for Dynamic Workloads. In *ICDE*. IEEE, 2196–2208.