Breaking It Down: An In-depth Study of Index Advisors

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

Chen Lin*
School of Informatics, Xiamen University
chenlin@xmu.edu.cn

Xuanhe Zhou*
Department of Computer Science and Technology, Tsinghua University
zhouxuan@tsinghua.edu.cn

Guoliang Li†
Department of Computer Science and Technology, Tsinghua University
liguoliang@tsinghua.edu.cn

ABSTRACT

Index advisors aim to improve workload performance by judiciously selecting an appropriate set of indexes. Various heuristic-based and learning-based methods have been proposed. However, there lacks a comprehensive assessment of existing index advisors, i.e., their advantages, limitations, and application scenarios. In this work, we conduct an in-depth study of existing index advisors in five key aspects. First, we initiate an end-to-end analysis, i.e., a completed analysis throughout the entire workflow of index advisors. We decompose index advisors into three essential building blocks, establish a taxonomy to classify methods used in each block, and analyze the strengths and weaknesses of these methods. Second, we develop a unified open-source testbed, implementing seventeen index advisors across eleven open-source or real-world datasets. We enable customizable configurations to meet diverse testing requirements. Third, we conduct an extensive assessment of index advisors across database systems in various scenarios. We evaluate their adaptability and robustness, identifying practical application scenarios. Fourth, we undertake a fine-grained ablation study by investigating variants of each building block. We identify effective variants and pinpoint significant factors impacting index advisors' performance via explainable machine-learning techniques. Lastly, we consolidate our findings that could shed light on research directions to advance the future development of index advisors.

PVLDB Reference Format:
Wei Zhou, Chen Lin, Xuanhe Zhou, and Guoliang Li. Breaking It Down: An In-depth Study of Index Advisors. PVLDB, 17(10): 2405 - 2418, 2024.
doi:10.14778/3675034.3675035

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/XMUDM/Index_EAB.

1 INTRODUCTION

Indexes are crucial for achieving high workload performance in databases [54]. Traditionally, indexes are identified, built, and maintained by the database administrator (DBA) [17, 45]. However, it is tedious for DBAs to manage indexes, especially for millions of database instances on the cloud [15, 65]. To automate this process, index advisors have been proposed [9, 11, 12, 27, 29, 39–41, 43, 53, 56, 64] to automatically and judiciously find an appropriate set of indexes to optimize the query performance.

There are two main categories of index advisors: (1) heuristic-based index advisors [9, 11, 12, 43, 53, 56] utilize predefined heuristic algorithms to create indexes; (2) learning-based index advisors [27, 29, 39–41, 64] build machine learning models based on training workloads and utilize these models to select indexes or estimate index benefits. These index advisors have different application scenarios, and *it is vital to evaluate their strengths and limitations for practical usage*. For instance, although learning-based methods are commonly believed to find near-optimal indexes for a given workload once well-trained [31], our experimental results show their performance is unpredictable, e.g., learning-based methods may perform worse than heuristic-based methods as the storage budget of indexes increases. Furthermore, existing works either evaluated heuristic-based index advisors in static scenarios (i.e., fixed workloads) and did not assess learning-based index advisors [26] or only investigated the robustness of index advisors [61, 62].

To tackle these limitations and assist database users in identifying the most suitable index advisors for their applications, we conduct an in-depth study of both the heuristic-based and the learning-based index advisors and make the following contributions.

❶ ***(End-to-End Analysis)*** **We provide a completed analysis encompassing the entire workflow of index advisors by constructing a taxonomy of an index advisor's building blocks and the methodology within each block (Section 2).** Although some heuristic-based index advisors are summarized in [26], there lacks a completed analysis in an end-to-end workflow for different index advisors (i.e., heuristic-based and learning-based methods). To acquire a clear understanding of the working mechanism of index advisors, we break down an index advisor into three essential building blocks (i.e., index candidate generation, index selection, and index benefit estimation), outline their typical methods, and summarize their strengths and weaknesses.

❷ ***(Unified Testbed)*** **- We develop a testbed that implements seventeen index advisors (and up to 144 variants) across eleven datasets with a unified interface (Section 3).** Existing studies developed index advisors independently [27, 39], did not provide open-source implementations [49, 58], or only focused on heuristic-based index advisors [26]. A testbed that accommodates both the heuristic-based and the learning-based index advisors is still missing. To fulfill the goal of a thorough assessment, we develop an open-source testbed. (1) Regarding the index advisors,

*The authors contribute equally
†Corresponding author

Proceedings of the VLDB Endowment, Vol. 17, No. 10 ISSN 2150-8097.
doi:10.14778/3675034.3675035

we implement both traditional methods widely used in real systems [11, 12, 53] and newly-developed learning-based index advisors [27, 39, 40, 64]; (2) Regarding the underlying datasets, we prepare testing suites based on multiple standard benchmarks [19, 32] and real-world datasets [63]; (3) We also support the evaluations of customizable combinations of various building blocks over a set of configurable parameters to meet different testing requirements (e.g., analytical and transactional queries with different patterns).

❸ ***(Comprehensive Evaluation)*** **- We place index advisors under different scenarios and conduct extensive experiments to evaluate their adaptability and robustness over both the open-source and real-world datasets (Section 4).** To the best of our knowledge, there is only a single study [26] that delved into the experimental evaluation of index advisors. However, this work only focused on the evaluation of heuristic-based index advisors. Besides, most previous studies synthesized workloads with limited and static query templates in OLAP benchmarks [29, 39], which fail to reflect the performance of index advisors in real scenarios. To cope with these limitations, we conduct comprehensive evaluations to test the capability of index advisors in three main scenarios, i.e., (1) static analytical scenarios, (2) dynamic analytical scenarios, and (3) transactional scenarios. For example, we generate workloads with small (e.g., varying the query frequency) to large changes (e.g., generating the query randomly) for assessing the robustness of index advisors in dynamic analytical scenarios.

❹ ***(Fine-grained Comparison)*** **- We explore different variants of index advisors, compare their performance, and utilize explainable machine learning techniques to reveal the underlying mechanism (Section 5).** Previous studies were restricted to the original implementations of index advisors [26]. As modern index advisors are getting more complex, it is vital to understand the causality and contribution of each building block of index advisors. Thus, we create multiple index advisor variants by exploring diverse combinations of the underlying building blocks. For example, we compare the performance of different filtering methods to generate index candidates and adopt explainable machine learning techniques [25] to identify the most influential features to predict useless indexes and filter them accurately.

❺ ***(Insightful Findings)*** **- We summarize findings from the evaluation results to shed light on the selection and future research of index advisors (Section 6).**

(1) Learning-based index advisors can identify more effective indexes than heuristic-based index advisors for static scenarios with the same queries.

(2) Learning-based index advisors with offline learning policy, which undergo an offline training stage across various workloads and directly return indexes for a new workload, are more efficient than others in most scenarios (especially for large datasets).

(3) Heuristic-based index advisors are more robust than learning-based index advisors for dynamic scenarios with workload drifts and data shifts.

(4) Index candidate generation can significantly affect the performance of index selection, i.e., preserving more index candidates typically contributes to better performance but higher time overhead. Empirical rules can enhance the effectiveness and efficiency of index candidate generation.

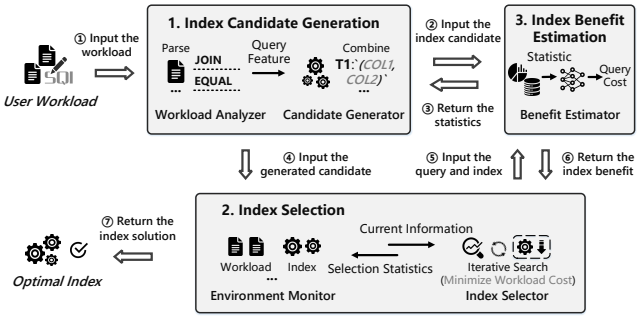

Figure 1: Workflow of Index Advisor

(5) The design of the input features for learning-based index advisors (i.e., the state representation in reinforcement learning) is critical to finding effective indexes.

(6) Tree-based estimation methods (e.g., XGBoost), utilizing query plan features (e.g., cardinality and costs) as input, achieve the highest accuracy in estimating index benefits.

2 INDEX ADVISOR

2.1 Workflow Overview

As shown in Figure 1, for a given workload, an index advisor generally contains three building blocks. First, **Index Candidate Generation** synthesizes promising index candidates using predefined strategies (Section 2.2). Then, **Index Selection** iterates over the generated index candidates and selects indexes based on the underlying selection mechanisms (Section 2.3). **Index Benefit Estimation** is leveraged to estimate the benefit of building an index to assist the aforementioned two building blocks. It provides estimated statistics for Index Candidate Generation to determine which index is an effective candidate. It quantifies the selection criteria to instruct Index Selection to choose an appropriate solution (Section 2.4).

In designing an index advisor, some key factors need to be considered. First, the working mechanism, i.e., whether the index advisor employs a learned component. Based on the working mechanism, index advisors can be classified into two categories: (1) the *heuristic-based* methods with a series of predefined algorithms; (2) the *learning-based* methods involving various learned components with different learning paradigms (e.g., reinforcement learning, classification, and regression). Second, the selection constraint, including (1) the *budget*, e.g., the storage of the index or the maximal number of indexes to be built; (2) the *types* of the index to be considered, e.g., single-column index or the multi-column index. Note that implementing an efficient physical index design [28] is not the focus of index advisors, which aim to select effective indexes. We utilize B-Tree as the default physical index design in this work.

Table 1 presents a list of existing index advisors investigated in this paper. We assess the representative index advisors including the ones widely utilized in the production systems and the ones newly emerged in recent studies. We next present a detailed analysis of each building block of these index advisors (e.g., the taxonomy, the strengths, and the weaknesses) in the following subsections.

2.2 Index Candidate Generation

Index candidate generation aims to prepare a set of effective candidates for index selection. As shown in Figure 2, there are four types

Table 1: Taxonomy and Characteristic of Index Advisors (#Index denotes the number of indexes, S/M denotes the single-column index and the multi-column index respectively, $\frac{Cost}{Storage}$ denotes the selection criteria that considers the size of the indexes)

Category	Index Advisor	Budget	Index Type	Index Candidate Generation	Index Selection (Criteria)	Index Benefit Estimation
Heuristic-based	Extend [43]	Storage	S/M	Prefix-based Expansion	Heuristic Search Strategy ($\frac{Cost}{Storage}$)	Statistic-based Method
	DB2Advis [53]	Storage	S/M	Random Column Permutation	Heuristic Search Strategy ($\frac{Cost}{Storage}$)	Statistic-based Method
	AutoAdmin [12]	#Index	S/M	Prefix-based Expansion	Heuristic Search Strategy (*Cost*)	Statistic-based Method
	Drop [56]	#Index	S	Random Column Permutation	Heuristic Search Strategy (*Cost*)	Statistic-based Method
	Relaxation [9]	Storage	S/M	Random Column Permutation	Heuristic Search Strategy ($\frac{Cost}{Storage}$)	Statistic-based Method
	DTA [11]	Storage	S/M	Random Column Permutation	Heuristic Search Strategy (*Cost*)	Statistic-based Method
	GUFLP [10]	Storage	S/M	Random Column Permutation	Mathematical Solver (*Cost*)	Statistic-based Method
	CoPhy [16]	Storage	S/M	Rule-based Construction	Mathematical Solver (*Cost*)	Statistic-based Method
Learning-based	SWIRL [27]	Storage	S/M	Random Column Permutation	Learned Selection Policy ($\frac{Cost}{Storage}$)	Statistic-based Method
	DRLindex [40]	#Index	S	Random Column Permutation	Learned Selection Policy (*Cost*)	Statistic-based Method
	DQN [36]	#Index	S/M	Rule-based Construction	Learned Selection Policy (*Cost*)	Statistic-based Method
	DBA Bandits [39]	Storage	S/M	Rule-based Construction	Learned Selection Policy (Cost)	Actual Runtime Statistics
	AutoIndex [64]	Storage	S/M	Rule-based Construction	Learned Selection Policy (*Cost*)	Learned Estimation Model
	BudgetMCTS [58]	#Index	S/M	Prefix-based Expansion	Learned Selection Policy (*Cost*)	Statistic-based Method
	AI Meets AI [20]	#Index	S/M	Prefix-based Expansion	Heuristic Search Strategy (*Cost*)	Learned Estimation Model
	DISTILL [49]	Storage	S/M	Learned Filter Model	Heuristic Search Strategy (*Cost*)	Learned Estimation Model
	ISUM [48]	Storage	S/M	Learned Filter Model	Heuristic Search Strategy (*Cost*)	Statistic-based Method
	LIB [47]	Storage	S/M	Random Column Permutation	Heuristic Search Strategy (*Cost*)	Learned Estimation Model
	QueryFormer [60]	Storage	S/M	Random Column Permutation	Heuristic Search Strategy (*Cost*)	Learned Estimation Model

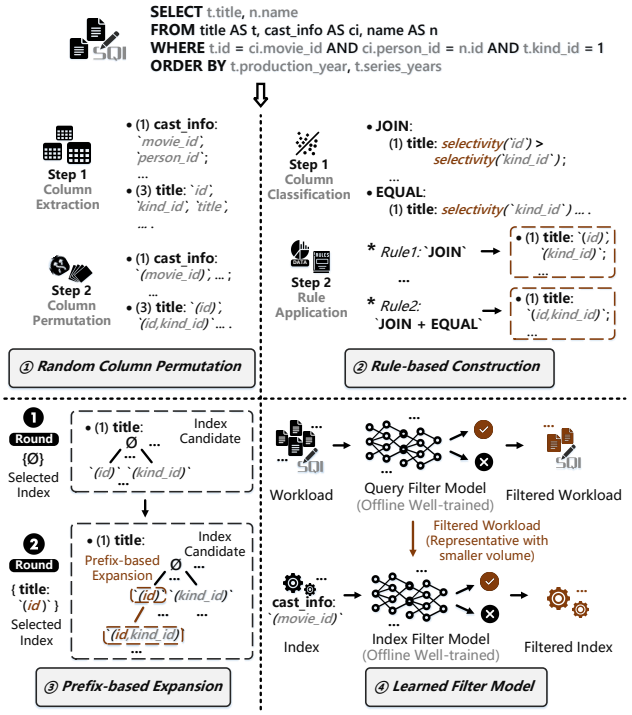

Figure 2: Index Candidate Generation Methods

of index candidate generation methods to alleviate searching over enormous candidate space [49].

❶ **Random Column Permutation.** The first type generates candidates in a two-step manner. As presented in Figure 2, it first parses and extracts all the indexable columns in the workloads. Next, it synthesizes index candidates by conducting permutations over the extracted columns within a given maximum index width (i.e., the number of columns in the index). To further reduce the volume of the candidates, some index advisors [40, 56] only generate single-column indexes, and others [9, 11, 53] leverage the what-if calls r to only preserve the indexes utilized in the query plans.

❷ **Rule-based Construction.** The second type generates candidates using rules that consider query patterns. As shown in Figure 2, index advisors [29, 39] first parse and classify the indexable columns into several groups based on the syntactic characteristics (e.g., JOIN and EQUAL denote the columns appear in join and equal predicates respectively). Then they utilize a series of heuristic rules to construct effective candidates (e.g., candidate 'title: (id, kind_id)' is generated according to '*Rule2*: JOIN + EQUAL' that combines columns in join and equal predicates). Apart from the syntactic features, some index advisors [34, 64] also take the data statistics of each column (e.g., the columns' selectivity: 'selectivity(id) > selectivity(kind_id)') into account during the generation process.

❸ **Prefix-based Expansion.** Instead of generating a fixed set of candidates at the beginning (i.e., a one-time effort), the third type maintains index candidates dynamically. It continuously refines the candidates based on the indexes selected in the previous rounds. As displayed in Figure 2, candidates in each round are generated by appending columns to the selected indexes (e.g., adding column 'kind_id' to the selected index '(id)' to be a new candidate '(id,kind_id)') [43]. The index width of the index candidates (i.e., the number of indexed columns) gradually increases, and a multi-column index candidate can only be considered when the candidate of its prefix has been selected [12, 27].

❹ **Learned Filter Model.** The last type employs learned models to filter out useless index candidates. As shown in Figure 2, there are two consecutive models. The Query Filter Model identifies representative queries in the workload. The model's inputs are various statistics (e.g., the table size, the column selectivity, and the query cost), and the model's outputs are the filtered representative queries in the workload. Then, index candidates are generated (i.e., permutate the indexable columns) based on the workload with these representative queries [48]. These index candidates are then fed to the Index Filter Model, which estimates the benefit of each index candidate based on index-related features (e.g., the statistics of each node in the query plan) and eliminates the useless ones (i.e., the impact is less than a given filtering threshold) [49].

Training and Inference. The learned filter model is workload-dependent and is trained with a labeled dataset, including the workload statistics. Once well-trained, it can be applied to compress the testing workloads or filter out useless index candidates. Training is necessary for each dataset, and retraining is required when the testing workloads differ significantly from the training workloads.

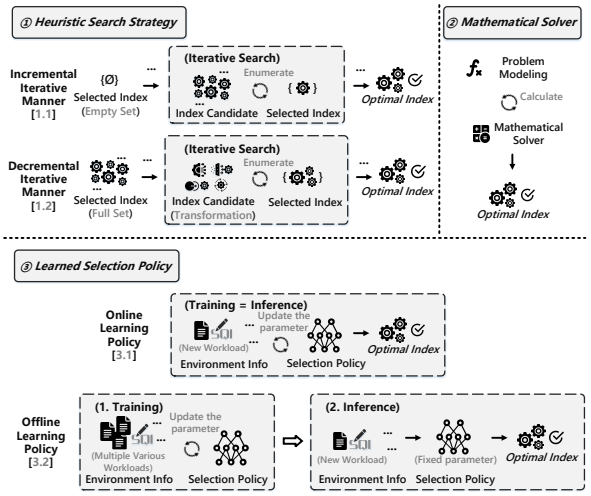

Figure 3: Index Selection Methods

2.3 Index Selection

Index selection explores the generated candidates and identifies the most effective indexes based on various strategies or algorithms. Generally, the selection is instructed by the formulated criteria, which either consider the pure cost (i.e., *Cost*) [11, 12] or the relative cost w.r.t. the size of the index (i.e., $\frac{Cost}{Storage}$) [43]. As shown in Figure 3, there are three types of selection methods.

❶ Heuristic Search Strategy. The first type determines indexes with different heuristic strategies gradually until the stop criterion is satisfied (e.g., the selected indexes exceed the storage budget). Based on the initial status of the selected index, there are two manners to conduct index selection. (1) *incremental*: add more indexes from an empty set; (2) *decremental*: remove indexes from a full index set.

► *Incremental Iterative Manner.* As displayed in Figure 3, this manner starts from an empty set and adds indexes to the selected index set incrementally. Specifically, most index advisors adopt greedy algorithms to choose the indexes that maximize the predefined selection criteria. Since the solely greedy strategy will likely return a sub-optimal indexing solution, some index advisors [12] adopt a mixed search strategy. Specifically, they first utilize the exhaustive search to derive several optimal indexes in a brute-force manner and then the greedy search to derive the remaining indexes. Another way to escape a sub-optimal solution is to alter indexes selected (e.g., randomly exchange some selected indexes with the ones not selected) and verify whether they are superior [53].

► *Decremental Iterative Manner.* As displayed in Figure 3, this manner initializes with all the candidates. It reduces the set of the selected indexes by discarding indexes until the given budget is satisfied. To acquire a reduced set of indexes (i.e., a smaller number or a lower storage budget), some index advisors [56] iteratively exclude the most "useless" index, e.g., removing the index that results in a smaller workload cost than removing other indexes. Others employ index transformations [9], including: (1) Merging: append the columns of one index to the end of another index. For example, it merges the index 'title: (id)' and the index 'title: (kind_id)' to get a new index 'title: (id, kind_id)'; (2) Splitting: split two indexes with shared prefix columns into three indexes. For example, it splits the two indexes ('title: (id, kind_id)' and 'title: (id,title)') into three indexes, i.e., the shared prefix ('title: (id)') and two indexes with the remaining columns ('title: (kind_id)' and 'title: (title)'); (3) Truncating: truncate the last column of an index. For example, it removes the column 'kind_id' in index 'title: (id, kind_id)' and gains a new index 'title: (id)'; (4) Removing: remove an index from the candidate set; (5) Promotion to a clustered index: transform a non-clustered index to a clustered index. The new clustered index could speed up range queries in the target workload and reduce the index size.

Table 2: Characteristics of Learned Selection Policy

Index Advisor	Learning Algorithm	State Represetation Workload	State Represetation Action	Action Space	Reward Function
SWIRL [27]	PPO (Offline Learning)	1. Query Representation 2. Query Frequency 3. Query Cost	1. Storage Info 2. Cost Sum Info 3. Index Status	Random Column Permutation	$\frac{Cost}{Storage}$
DRLindex [40]	DQN (Offline Learning)	1. Query Representation 2. Accessed Column 3. Column Selectivity	Index Status	Random Column Permutation	*Cost*
DQN [29, 35]	DQN (Online Learning)	Query Frequency	Index Status	Rule-based Construction	*Cost*
DBA Bandits [39]	MAB (Online Learning)	N/A	Index Status	Rule-based Construction	*Cost*
AutoIndex [64]	MCTS (Online Learning)	N/A	Index Status	Rule-based Construction	*Cost*
BudgetMCTS [58]	MCTS (Online Learning)	N/A	Index Status	Prefix-based Expansion	*Cost*

❷ Mathematical Solver. The second type models index selection as a linear programming problem (LP) [7]. The choice of indexes is treated as binary variables, and the overall workload cost is a linear objective function based on the workload statistics (i.e., the workload cost) and the choice of indexes. An off-the-shelf mathematical solver derives the solution (i.e., the indexes) that minimizes the objective function subject to a set of constraints (e.g., the storage budget) [16, 42]. Moreover, to reduce the complexity of the problem, some index advisors restrict the solution space by only allowing a single index for each query [10]. Others decompose the original problem into several smaller ones and solve these smaller problems simultaneously to enhance the selection procedure [44].

❸ Learned Selection Policy. The third type transforms the selection procedure into a Markov Decision Process (MDP) [14] and utilizes reinforcement learning to solve this problem. Specifically, with *state representation* (information like workload characteristics) as the input, these methods iterate over a predefined *action space* (index candidates) via a learned selection *policy* (to decide an action of an index candidate) and refine the policy based on the *reward function*. Based on whether these methods can adapt to dynamic workloads, they can be classified into (1) *online learning* policy or (2) *offline learning* policy. The details about the MDP modeling of different methods are summarized in Table 2.

► *Online Learning Policy.* As displayed in Figure 3, this method learns to identify optimal indexes for a static workload through iterative trial-and-error attempts, where the selection overhead refers to the online training effort over the given workload. It captures the environment information (i.e., the workload characteristics and the current index status) at a coarse granularity. The workload is modeled by a one-dimensional query frequency vector where each channel corresponds to an individual query [29, 35]. The index candidates are modeled as a multi-hot index vector where each channel indicates whether the index candidate has been selected (e.g., 0 denotes not selected). The order of the columns in the index and the derived statistical information (e.g., the estimated size of the index) are also taken into account in some index advisors [39]. As opposed to using vector representations for indexes, some index advisors [58, 64] maintain a selection tree where each tree node

corresponds to a valid index combination. To optimize the policy, different learning algorithms are adopted, including Deep Q Network (DQN) [36] and Multi-armed Bandits (MAB) [39], which train a Neural network to represent the action policy and Monte Carlo Tree Search (MCTS) [8] which computes optimal actions through numerous simulations (i.e., without a neural network).

► *Offline Learning Policy.* Instead of conducting index selection for a static workload, this method handles dynamic workloads. The selection overhead only includes the inference overhead, facilitated by an offline training step. As shown in Figure 3, it first undergoes an offline training process across various workloads. Then, the learned policy directly returns indexes for a new workload. Specifically, this method models environmental information in a fine-grained manner. These methods characterize a workload with a query representation matrix, where each row is the representation generated by learned models [27] or filled with statistic vectors (e.g., the number of accessed column occurrences in the query) [40]. Some index advisors [40] also introduce other features like column selectivity. These methods leverage a multi-hot vector to reflect the status of current indexes. Some index advisors [27] also incorporate meta information (e.g., the storage budget). The selection policy is learned via Deep Q Network (DQN) [36] that estimates the Q-value (i.e., the index benefit) of each index and Proximal Policy Optimization (PPO) [46] that adjusts the parameters of the policy network to maximize the rewards (i.e., the index benefit).

Training and Inference. The learned selection policy is workload-dependent and is trained over a dataset with various workloads. For *online learning policy*, the training and the testing dataset contain the same workload. It only learns to identify indexes for the given workload and requires retraining when a new workload arrives (i.e., online learning process). For *offline learning policy*, the training and the testing datasets are different workloads. It directly returns indexes for the testing workload with the knowledge (e.g., the index-related query patterns) acquired over the training workloads (i.e., offline learning process). It undergoes training for each dataset and requires retraining when significant query drift occurs.

2.4 Index Benefit Estimation

Index benefit estimation aims to estimate the benefits of utilizing the selected indexes without actually building the indexes (i.e., the actual runtime statistics), which includes two main types.

❶ **Statistic-based Method.** The first type utilizes the estimated statistics from the database optimizer to calculate the index benefits. It derives the estimated statistics via the what-if calls of the database optimizer with the usage of hypothetical indexes (e.g., the database optimizer equipped with the HypoPG [1] extension in PostgreSQL [3]). Specifically, the what-if calls return (1) the query cost by an empirical cost function about the weighted sum of different operations (e.g., seq_page_cost) [57] and (2) the index size by a predefined size function based on the estimated statistics (e.g., the tuple_num and the page_size) [1].

❷ **Learned Estimation Model.** The second type employs learned models to estimate index benefit. These models are trained to serve as a mapping function from the input workload statistics and indexes to the value that reflects its benefit (e.g., the relative cost reduction [47]). Unlike learned cardinality or cost estimation models [50, 51], which require only workload features (e.g., the joined tables), learned index benefit estimation models require features that characterize the index (e.g., the selectivity of the columns in the index) to estimate its benefit. Overall, there are two categories:

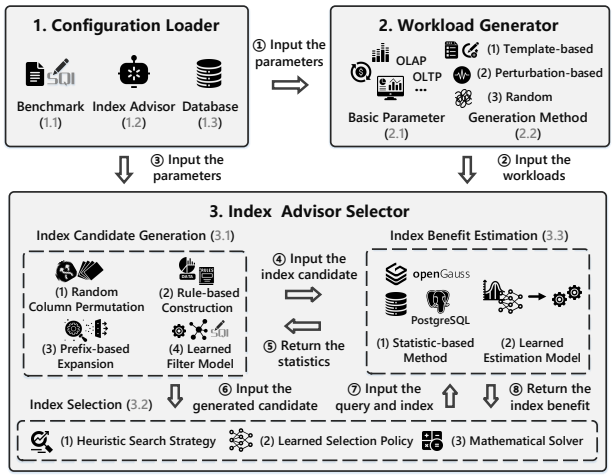

Figure 4: Workflow of the Proposed Testbed

► *Classification Model.* The first model takes estimation as a classification problem and predicts which of the two input indexes is more effective. Given two indexes and the corresponding query plans, [20] trains a classifier to return the index with a larger index benefit based on the well-defined feature vectors of the query plan (e.g., the sum of the estimated cost of each node).

► *Regression Model.* The second model treats estimation as a regression problem and estimates the workload cost ($cost_{w/\ index}$) or its reduced ratio ($1-\frac{cost_{w/\ index}}{cost_{w/o\ index}}$) [47, 49] over the given indexes. They propose deep learning models (e.g., the attention-based model [47]) to characterize index-related features for an accurate estimation. For example, they utilize the operation information (e.g., the operator type affected by the index in the query plan), the database statistics (e.g., the distinct ratios of the indexed columns), and the index information (e.g., the order of the columns in the index) to estimate index benefit [47, 60].

Training and Inference. The learned estimation model is workload-dependent and predicts index benefits given the index-related features. Specifically, it undergoes an offline training process with a labeled dataset, including the workload features (e.g., the query plan) and the actual index benefits (e.g., the reduced query latency). Then, it can be applied to estimate the index benefit given the testing workloads and the indexes. It needs to be trained once for each dataset and requires retraining when significant query drift occurs.

3 EXPERIMENTAL TESTBED

We develop the first open-source testbed that (1) includes heuristic-based and learning-based index advisors with 17 methods in total, (2) supports a fine-grained comparison over the building blocks of index advisors with up to 144 variants, and (3) provides a unified interface to conduct evaluation over 11 datasets with various user-specified parameters in different scenarios. As shown in Figure 4, the testbed includes three modules.

► **Configuration Loader.** This module initializes a series of evaluation settings, which involve the following aspects. **(1)** ***Benchmark:***

the datasets (e.g., the OLAP benchmark TPC-H [5]), the query parameters (e.g., the number of queries), and the workload generation methods (e.g., query template-based generation); **(2) *Index Advisor:*** the assessed index advisors, the methods of the three building blocks, and the selection constraints (e.g., the storage budget); **(3) *Database:*** the target databases, i.e., PostgreSQL and openGauss, that support hypothetical indexes [1].

► **Workload Generator.** This module supports three methods for generating workloads with diverse features (e.g., query changes due to typical workload drifts [33]) to simulate the requirements posed by various scenarios. **(1) *Template-based Method*** generates queries by filling placeholders in the predefined query templates with different values (e.g., the 22 templates in TPC-H benchmark); **(2) *Perturbation-based Method*** augments queries by applying a series of perturbations (i.e., minor query changes). It simulates the workload drifts due to daily transactions or user behavior [19, 54] by perturbing the given queries [61], such as (*a*) modifying values in the filter predicates, (*b*) adding additional columns in the SELECT clause, and (*c*) adjusting the order of the columns in the ORDER BY clause; **(3) *Random Method*** synthesizes Select-Project-Join (*SPJ*) queries based on the PrimaryKey-ForeginKey (PK-FK) relations (i.e., the specified join predicates) with varying numbers of randomly generated filter predicates [24, 51, 55, 59]. More details about the workloads are in the experimental settings of each section.

► **Index Advisor Selector.** This module implements index advisors listed in Table 1, including seven heuristic-based index advisors and ten learning-based index advisors. **(1) *Heuristic-based*:** We refactor the implementations of heuristic-based index advisors in [26] (e.g., extend the selection constraints to include both the storage budget and the maximum allowable number). We implement index advisors with the selection policy of mathematical solver (e.g., CoPhy [16]); **(2) *Learning-based*:** We implement index advisors that employ learned components in the three building blocks. For Index Candidate Generation, we implement DISTILL [49] with multiple tree-based variants (e.g., XGBoost [13]). For Index Selection, we implement DRLindex [40] with the proposed state representation (i.e., the query representation matrix, the column selectivity vector, the assessed column vector, and the multi-hot index status vector). We implement DQN [29, 35] by leveraging the state representation in SWIRL [27] (i.e., the query representation, the query frequency/cost vector, the multi-hot index status vector, the cost sum, and the storage budget) to support dynamic workloads. For Index Benefit Estimation, we implement the classifier of tree-based models with the statistical features (i.e., the query cost and cardinality) and model it as a regression problem (i.e., a cost estimator) [20]. We refactor the implementations of LIB [47] and QueryFormer [60] to support workloads across datasets.

Benchmark Dataset. We employ various open-source benchmarks and real-world datasets for evaluating performance in the OLAP or OLTP scenario. For the OLAP scenario, we adopt five benchmarks with different characteristics and complexities. (1) TPC-H [5]: an open-source decision support benchmark, where the data is uniformly distributed with a small schema of 22 query templates and 61 columns; (2) TPC-DS [4]: an open-source decision support benchmark, which is complex with a large schema of 99 query templates and 429 columns; (3) JOB [32]: a real-world IMDB dataset with 113 query templates and 108 columns, where the overuse of indexes leads to performance regression; (4) TPC-H Skew [39]: an augmented version of TPC-H with a skewed data distribution of 22 query templates and 61 columns; (5) DSB [19]: an enhanced version of TPC-DS with 52 more complex query templates and 429 columns. For the OLTP scenario, we create multiple TPC-C benchmarks [18] with varying Read/Write ratios by adjusting the proportions of the five transactions, along with multiple real-world datasets containing transaction queries [63]. Note that our testbed can add other datasets with little effort (e.g., specify the database connection).

Table 3: Comparison of Learning-based Methods (RL denotes reinforcement learning and SL denotes supervised learning)

Index Advisor	Model Type	Model Size (MB)	#Parameters	Training Time (s)	Training Paradigm
SWIRL [27]	Network	3.58	927,065	5,090	RL
DRLindex [40]	Network	1.17	300,527	985	RL
DQN [29]	Network	1.12	281,891	1,467	RL
AI Meets AI [20]	Tree-based	26.43	N/A	44	SL
DISTILL [49]	Tree-based	3.03	N/A	17	SL
LIB [47]	Network	0.39	9,1617	2,343	SL
QueryFormer [60]	Network	11.95	3,118,167	3,360	SL

Evaluation Metric. (1) Relative Cost Reduction: the benefit of the indexes, i.e., the ratio of reduced workload cost (i.e., $1-\frac{cost_{w/\ index}}{cost_{w/o\ index}}$) returned by various index advisors; (2) Time Duration: the time overhead required to get the indexes returned by different index advisors. For learning-based index advisors with offline learning policy (e.g., SWIRL), their training overhead is exclusively considered and reported beyond this metric; (3) Query Latency: the actual query runtime with the selected indexes; (4) Throughput: the number of queries or transactions processed within a given time horizon with the selected indexes; (5) Q-Error: the estimation accuracy (i.e., $max(\frac{benefit_{act}}{benefit_{est}}, \frac{benefit_{est}}{benefit_{act}})$) of the index benefit estimation models; (6) F1 Score: the classification performance [31] of the learned filter model in identifying effective index candidates.

Implementation. Experiments are conducted with Python 3.7, PostgreSQL 12.5, and openGauss 5.0.0 on a workstation with two Intel (R) Xeon (R) CPU E5-2678 v3 @ 2.50GHz, 256 GB main memory, and 4 GeForce RTX 2080 Ti graphics cards. The default selection constraint is the storage budget, set at half the size of the dataset (e.g., 500MB for a 1GB dataset), and the default maximum index width (the number of used columns) is 2. For learning-based index advisors, the workloads are split into training/validation/testing set by 8:1:1. The hyperparameters are determined by employing the values from the original implementations or utilizing grid search and cross-validation methods. The offline training overhead for the offline learning policy is set to be 100,000 epochs by default. We compare important training parameters of learning-based methods in Table 3. For example, *Training Time* displays the time spent to train the model within a given budget (e.g., the same predefined epochs for methods with the same *Model Type* and *Training Paradigm*). The results are derived over three runs to alleviate the impact of external factors (e.g., the fluctuation in resource availability).

4 OVERALL PERFORMANCE EVALUATION

In this section, we evaluate the overall performance of index advisors across different scenarios. Our investigation delves into their performance over workloads with diverse features (e.g., analytical or transactional queries). We also consider the selection granularity, i.e., delivering indexes for a query or a workload.

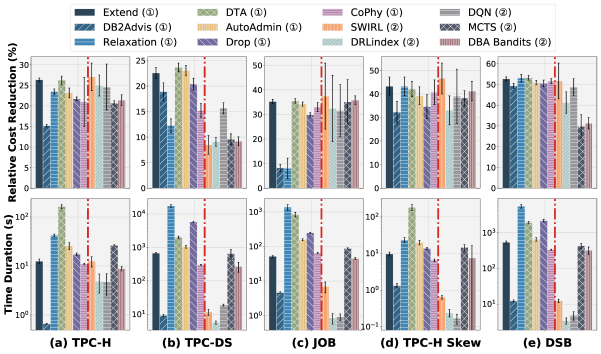

Figure 5: Workload-level Performance (① denotes heuristic-based index advisor; ② denotes learning-based index advisor)

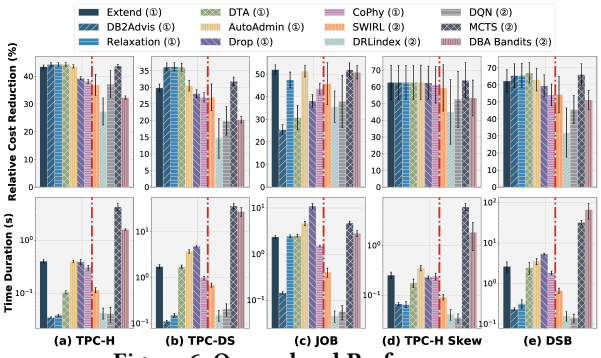

Figure 6: Query-level Performance

Table 4: Training Overhead of Learning-based Index Advisors

Method	TPC-H	TPC-DS	JOB	TPC-H Skew	DSB
SWIRL [27]	1h 24m 50s	7h 46m 47s	6h 11m 7s	51m 51s	5h 35m 3s
DRLindex [40]	**16m 25s**	**2h 39m 25s**	**22m 32s**	**14m 17s**	**55m 22s**
DQN [29]	24m 27s	4h 13m 50s	43m 43s	23m 53s	3h 1m 21s

Table 5: Time Duration (s) of Each Building Block

Method	TPC-H			TPC-DS		
	Index Candidate Generation + Index Selection	Index Benefit Estimation		Index Candidate Generation + Index Selection	Index Benefit Estimation	
		Simulation	Cost Request		Simulation	Cost Request
Extend [43]	1.01 (8.1%)	0.33 (2.6%)	**11.16 (89.3%)**	58.6 (9.5%)	3.84 (0.6%)	**554.48 (89.9%)**
DB2Advis [53]	0.15 (22.7%)	0.25 (37.9%)	**0.26 (39.4%)**	1.77 (19.7%)	2.39 (26.7%)	**4.8 (53.6%)**
Relaxation [9]	7.85 (18.6%)	1.76 (4.2%)	**32.58 (77.2%)**	6076.14 (36.3%)	92.31 (0.5%)	**10574.86 (63.2%)**
DTA [11]	13.95 (8.5%)	6.08 (3.7%)	**143.4 (87.8%)**	176.1 (8.8%)	14.82 (0.7%)	**1824.83 (90.5%)**
AutoAdmin [12]	1.49 (5.8%)	1.54 (6.0%)	**22.68 (88.2%)**	45.87 (4.1%)	21.68 (2.0%)	**1037.37 (93.9%)**
Drop [56]	7.73 (44.5%)	1.12 (6.4%)	**8.54 (49.1%)**	**5282.8 (88.0%)**	29.27 (0.5%)	689.62 (11.5%)
CoPhy [16]	0.79 (7.2%)	0.30 (2.7%)	**9.85 (90.1%)**	14.57 (4.8%)	2.12 (0.7%)	**286.83 (94.5%)**

– **RQ 4.1:** How effectively can index advisors handle complex analytical queries in static analytical scenarios? Furthermore, how about their performance over workload-level and query-level selection?
– **RQ 4.2:** To what extent can index advisors adapt to dynamic analytical scenarios with workload drifts? For example, how well do they handle minor changes (e.g., similar query templates with varying query frequency) and extreme changes (e.g., random queries)?
– **RQ 4.3:** How effectively can index advisors manage the transactional workloads, including the IUD statements (i.e., INSERT, UPDATE, and DELETE statements) that induce data shifts?

4.1 Static Analytical Scenarios

Experimental Setting. We construct workloads based on the OLAP benchmark in Section 3, where the same query templates are populated with different parameter values via distinct random seeds. The workload size varies from one (i.e., query-level selection) to the number of templates in the benchmark (i.e., workload-level selection). The query frequency is randomly assigned by a value in [1, 1000]. The training and the testing workloads for all the learning-based methods are the same workloads. The results are the average performance of index advisors over these workloads.

O1 (Selection Effectiveness): Learning-based index advisors can identify more effective indexes but with a higher performance variance than heuristic-based index advisors over static workloads of same queries. As displayed in Figure 5, learning-based index advisors return more effective indexes than heuristic-based index advisors when the training and the testing workloads are comprised of the same queries. For example, SWIRL achieves a higher Relative Cost Reduction, i.e., 37.48% than 26.30% of heuristic-based index advisors over JOB benchmark on average. However, learning-based index advisors exhibit a higher performance variance, i.e., the standard deviation of the Relative Cost Reduction is 5.01 (2.55× of heuristic-based index advisors). It might be attributed to the fact that the learning algorithms enable learning-based index advisors to capture various workload patterns essential for selecting effective indexes via substantial trial-and-error attempts. However, the trial-and-error strategy might involve incorrect attempts that mislead the training process, introduce significant uncertainty regarding the quality of the learned policy, and lead to performance oscillation of learning-based index advisors.

On the contrary, heuristic-based index advisors return indexes with a smaller performance variance than learning-based index advisors. Among them, DTA [11] outperforms others with the Relative Cost Reduction, i.e., 36.14% on average. It might be attributed to the fact that DTA compares and considers candidates effective for a single query in the workload, which are likely to have a larger impact on the workload. Moreover, it supports the selection of multi-column indexes that accelerate the retrieval of multiple conjunctive filter predicates without restricting the width of index candidates (i.e., the number of columns) at the beginning [12].

O2 (Selection Efficiency):Learning-based index advisors with offline learning policy are more efficient than heuristic-based index advisors and learning-based index advisors with online learning policy. As shown in Figure 5, learning-based index advisors with offline learning policy (i.e., SWIRL [27], DRLindex [40] and DQN [29]) require less overhead[1], i.e., 591.39ms, 163.32× faster than others on average. The reason is that they return indexes without the time-consuming search process (i.e., enumerate and compare different indexes) and the online training procedure (i.e., update the policy network). Among them, DRLindex requires the shortest selection time 301.79ms, which is 1.44× faster than others. It might stem from: (1) the absence of maintaining index candidates dynamically in SWIRL; (2) a smaller set of index candidates (i.e., only single-column index) to select from than DQN. These also contribute to less training overhead for DRLindex. As shown in Table 4, DRLindex requires the least training overhead, i.e., 3216.2s of 100 thousand epoch (2.43× faster than SWIRL and DQN on average).

On the contrary, heuristic-based index advisors with decremental iterative manner are the most time-consuming, especially over large datasets. For example, the time overhead of Relaxation and Drop is 10.26× than other heuristic-based index advisors over TPC-DS and DSB benchmark on average. The underlying reason is that the decremental iterative manner includes all the index candidates initially. It requires more steps to remove indexes of larger datasets than the incremental iterative manner. To further identify the most time-consuming step of heuristic-based index advisors, we calculate the time distribution over each building block of index advisors in Table 5. We observe that the time spent on Index Benefit Estimation

[1]Note the time overhead of the offline training process is not included in time duration, which is reported in Table 4.

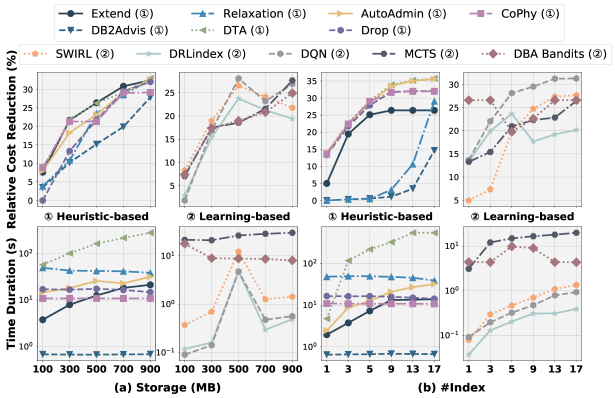

Figure 7: Performance w.r.t. Varying Budget Constraints

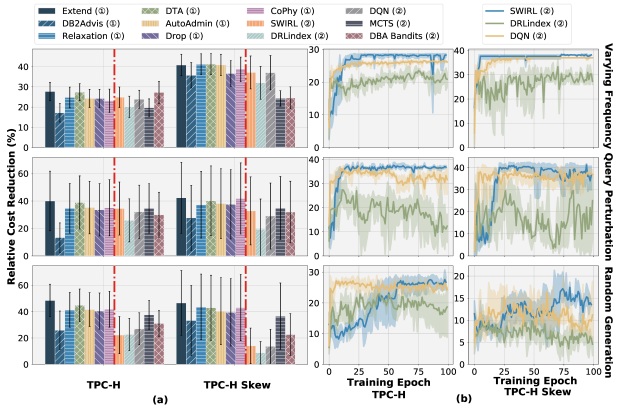

Figure 8: Relative Cost Reduction of (a) index advisors w.r.t. different workload changes and (b) learning-based index advisors with offline learning policy at each training epoch.

occupies a large proportion (i.e., 79.53% on average). It involves the usage of the what-if calls to simulate hypothetical indexes (i.e., Simulation) and derive estimated query costs (i.e., Cost Request) to calculate index benefits. Cost Request is the most time-consuming step, which occupies 72.73% of the selection time on average.

O3 (Selection Granularity): Index advisors perform better over the query-level selection than the workload-level selection. In contrast to the results in Figure 5, index advisors obtain higher Relative Cost Reduction over query-level selection as shown in Figure 6. Specifically, the respective values of heuristic-based and learning-based index advisors are 48.11% and 41.47% over query-level selection compared with 31.82% and 21.13% over workload-level selection. The time overhead of heuristic-based index advisors is also within an acceptable range, i.e., $< 10s$. DB2Advis is even more efficient than SWIRL across all the benchmarks. It indicates that a simple workload (i.e., only a single query) alleviates the difficulty of index selection (e.g., a smaller index space to explore). Thus, index advisors are expected to handle it well, regardless of their diverse internal designs.

O4 (Selection Budget): Heuristic-based index advisors identify more effective indexes given a larger storage budget, while learning-based index advisors might return inferior indexes when the storage budget increases. As displayed in Figure 7, the Relative Cost Reduction of heuristic-based index advisors shows an increasing tendency as the value of Storage and #Index increases. For example, the Relative Cost Reduction of DTA rises from 8.61% to 32.95% when Storage increases from 100MB to 900MB, and from 14.02% to 35.73% when #Index increases from 1 to 17. However, this is not the case for learning-based index advisors. As shown in Figure 7, the Relative Cost Reduction of SWIRL and DRLindex decline when the storage increases from 500MB to 900MB (i.e., from 26.53% to 21.75% for SWIRL and 23.75% to 19.40% for DRLindex). It aligns with the discovery of learned cardinality estimation models in [23, 55] that the estimated cardinality increases for queries with an additional predicate. Such unpredictability of learning-based methods raises concerns about their reliability. The lack of interpretability renders them inappropriate for real-world applications with rigorous requirements [6, 37].

4.2 Dynamic Analytical Scenarios

Experimental Setting. We construct workloads (split into the training/validation/testing set by 8:1:1 for learning-based index advisors) with different degrees of workload drifts (i.e., query changes) to verify the robustness of index advisors (i.e., the ability to maintain stable performance over dynamic workloads). (1) Varying Frequency: queries in the workloads remain the same but are assigned by a random frequency in $[1, 1000]$; (2) Query Perturbation: a series of perturbations (i.e., add a new predicate) are applied to queries in the workloads to simulate typical workload drifts [33, 61] introduced in Section 3; (3) Random Generation: queries in the workloads are randomly synthesized through the method (e.g., varying numbers of selection predicates based on the specified join predicates) in [24, 59]. It simulates extreme workload changes due to large workload drifts and poses large requirements for the robustness of index advisors.

O5 (Varying Frequency & Query Perturbation): Heuristic-based and learning-based index advisors are robust over workloads with varying frequency and small perturbations. As shown in Figure 8 (a), learning-based and heuristic-based index advisors achieve similar Relative Cost Reduction over workloads of Varying Frequency and Query Perturbation over TPC-H benchmark. The respective Relative Cost Reduction is 28.53% and 27.29% on average. However, the performance is significantly worse, i.e., 38.58% and 30.31% on average over TPC-H Skew benchmark with a skewed data distribution. We plot the training curves of learning-based index advisors with offline learning policy in Figure 8 (b). We observe that their training curves show a higher degree of oscillation for workloads of Query Perturbation over TPC-H Skew benchmark than workloads of Varying Frequency over TPC-H benchmark. All of these indicate the capability of learning-based index advisors to handle dynamic workloads of slight changes over simple data distributions since they can capture essential query patterns and identify effective indexes within a few trial-and-error attempts.

O6 (Random Generation): Learning-based index advisors are less robust than heuristic-based index advisors, and they converge poorly over random workloads. As shown in Figure 8 (a), learning-based index advisors obtain lower Relative Cost Reduction than heuristic-based index advisors over Random Generation workloads. The Relative Cost Reduction of learning-based index advisors is 28.10% compared with 40.61% of heuristic-based index advisors over TPC-H benchmark on average. For TPC-H Skew benchmark, the Relative Cost Reduction is 19.12% of learning-based index advisors against 41.31% of heuristic-based index advisors. Besides having lower effectiveness, learning-based index advisors converge

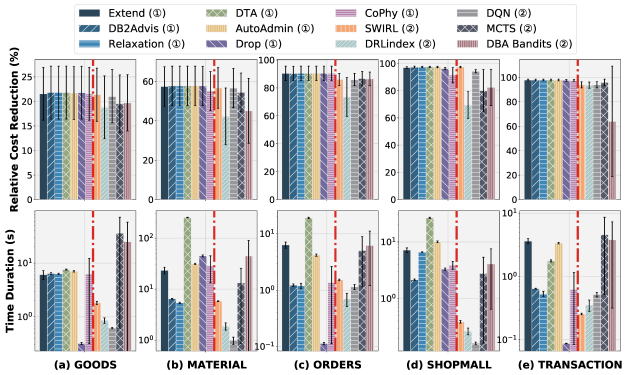

Figure 9: Performance over Simple Transactional Query

Table 6: Performance over TPC-C Benchmark Variants

Datasets	DEFAULT		READ HEAVY		WRITE HEAVY	
Method	Throughput (requests / s)	Latency (ms)	Throughput (requests / s)	Latency (ms)	Throughput (requests / s)	Latency (ms)
Heuristic-based Index Advisor						
Extend [43]	194.95	5115.40	689.48	1444.20	220.99	4513.60
DB2Advis [53]	201.25	5009.20	204.70	4932.80	220.37	4526.00
Relaxation [9]	203.73	4943.60	191.25	5219.80	221.75	4498.20
DTA [11]	**213.62**	**4669.40**	205.06	4871.20	221.42	4504.40
AutoAdmin [12]	192.88	5177.00	**708.84**	**1407.00**	221.30	4506.60
Drop [56]	48.61	20588.00	12.50	80457.00	148.20	6751.60
CoPhy [16]	210.70	4738.20	0.77	9121877.80	**223.30**	**4468.40**
Learning-based Index Advisor						
SWIRL [27]	1.21	1528165.93	148.94	21142.47	3.55	1326983.13
DRLindex [40]	7.48	271020.00	10.42	664973.93	9.87	1319985.93
DQN [29]	37.67	44955.53	108.85	14266.67	105.24	20294.20
MCTS [58, 64]	9.61	116962.00	102.07	34237.40	49.06	20363.00
DBA Bandits [39]	34.14	96717.20	114.49	18338.60	206.74	4820.60

poorly over Random Generation workloads. According to the training curves in Figure 8 (b), index advisors exhibit poor convergence performance, especially over TPC-H Skew benchmark. Random workloads encompass diverse query patterns, which poses a significant challenge to learning a well-behaved selection policy. Thus, index advisors require extensive trial-and-error attempts to find effective indexing solutions.

4.3 Transactional Scenarios

Experimental Setting. We evaluate the performance of index advisors over transactional workloads and investigate the impact of data shifts [33, 38] incurred by the IUD statements (i.e., INSERT, UPDATE and DELETE statements). (1) For transactional workloads, we conduct experiments over the real-world dataset and the open-source benchmark. Regarding the real-world dataset, we utilize five datasets introduced in [63] to assess their performance in managing simple transactional queries. These datasets possess different complexities with 750 to 1265 columns. Regarding the open-source benchmark, we vary the configurations in OLTPBench [18] to construct multi-version TPC-C benchmarks. We adjust the proportions of the five transactions (NewOrder, Payment, OrderStatus, Delivery, and StockLevel) in TPC-C benchmark to construct workloads with different Read/Write ratios. Specifically, we construct three TPC-C variations, i.e., DEFAULT, READ HEAVY, and WRITE HEAVY. DEFAULT corresponds to the standard transaction proportions in TPC-C benchmark. WRITE HEAVY focuses on workloads of high write ratio with two transactions (NewOrder and Payment). READ HEAVY only involves SELECT statements of two transactions (OrderStatus and Stock Level); (2) For data shifts, we assess the capability of index advisors to handle data shifts by manipulating the data volume of various benchmarks, e.g., learning-based index advisors are trained on a 1GB dataset but tested on a 10GB dataset.

O7 (Transactional Workload): Index advisors perform well over transactional workloads with simple SELECT statements. As shown in Figure 9, index advisors achieve comparable Relative

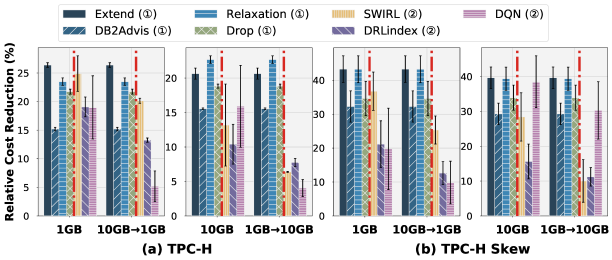

Figure 10: Performance over Data Shift (1GB → 10GB denotes that learning-based index advisors are trained on a 1GB dataset but are tested on a 10GB dataset)

Cost Reduction across five datasets, e.g., 72.71% for heuristic-based index advisors and 65.57% for learning-based index advisors on average. Furthermore, the Time Duration is typically low, i.e., $< 100s$ across datasets. The underlying reason might be attributed to the characteristics of transactional queries, which are Select-Project-Join (*SPJ*) queries from predefined templates, with fewer selection predicates (e.g., the number of the selection predicates is only 3.5 across the five datasets on average) and without multiple nested subqueries of complex analytical queries in Section 4.1. The simple characteristics lead to fewer index candidates, alleviating the difficulty in identifying effective indexes.

O8 (Varying Read/Write Ratio): Heuristic-based index advisors perform well over READ HEAVY workloads, and learning-based index advisors are not effective over workloads including IUD statements. As shown in Table 6, heuristic-based index advisors can achieve good performance over READ HEAVY workloads. For example, Extend and AutoAdmin outperform other index advisors with the Throughput, i.e., 699.16 $request/s$ on average (5.36× larger than others) and the Latency, i.e., 1425.6ms on average (698.38× smaller than others). However, their superiority diminishes over DEFAULT or WRITE HEAVY workload of high write ratios. For example, all the heuristic-based index advisors other than Drop achieve a roughly 200 $request/s$ of Throughput over these two workloads. Meanwhile, learning-based index advisors hardly outperform the heuristic-based index advisors, especially over DEFAULT or WRITE HEAVY workloads with IUD statements. For instance, SWIRL demonstrates an extremely low Throughput, i.e., 1.21 $request/s$ and 3.55 $request/s$ on DEFAULT and WRITE HEAVY workload, respectively. The underlying reason might be attributed to the fact that existing index advisors only consider the benefit of indexes for the SELECT statements during the selection process. They ignore the overhead of the IUD statements, e.g., the index maintenance cost associated with updating the underlying index structure. Thus, index advisors are unable to account for the negative impact induced by the IUD statements and exhibit poor performance when handling workloads with a large volume of the IUD statements (i.e., a high write ratio).

O9 (Data Shift): Heuristic-based index advisors are robust over data shift, while learning-based index advisors undergo performance regression when data shift occurs. As shown in Figure 10, heuristic-based index advisors showcase consistent performance when the volume of the underlying data shifts between 1GB and 10GB. However, learning-based index advisors encounter performance regression when a data shift occurs. For example, SWIRL experiences a 14.89% decrease of Relative Cost Reduction over TPC-H Skew benchmark on average. This phenomenon might be attributed to the difference in the working mechanism of heuristic-based index advisors and learning-based index advisors.

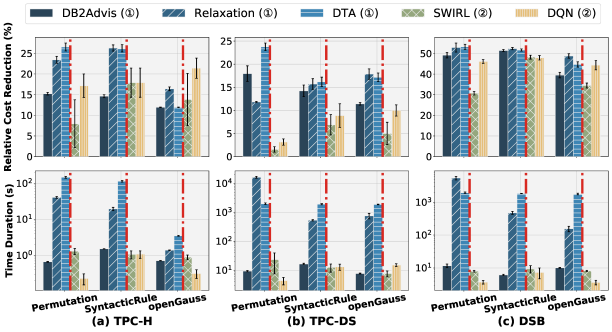

Figure 11: Performance of Candidate Generation Variants

Heuristic-based index advisors directly conduct index selection from scratch for a workload over any given data distribution. However, learning-based index advisors return indexes with the learned policy acquired in an offline process, which is tailored for specific data distribution. Thus, the learned policy does not apply to the changed data distribution, and learning-based index advisors encounter performance regression due to the mismatch between the learned policy and the new data distribution. We further investigate the impact of data shifts by training learning-based index advisors with two training manners, i.e., training from scratch without any knowledge of the previous dataset and continuous retraining with the acquired knowledge. The experimental results can be find at [2].

5 INFLUENCE OF BUILDING BLOCKS

In this section, we investigate the impact of the three building blocks of index advisors summarized in Section 2. We construct and compare the performance of various index advisor variants with different combinations of the underlying building blocks, which are assessed over workloads with the same generation methods in Section 4.1. The testing workloads of learning-based index advisors are the same query templates as their training workloads but might be filled with different parameter values.

– **RQ 5.1:** What is the most effective candidate generation method?
– **RQ 5.2:** What are the most influential factors in index selection? How do they influence the selection of effective indexes?
– **RQ 5.3:** What is the most accurate index benefit estimation model? How do they affect the end-to-end workload performance?

5.1 Index Candidate Generation

Experimental Setting. According to Table 1, there are four types of index candidate generation methods; three of them are heuristic-based, and one is learning-based (i.e., learned filter model). On the one hand, we assess the end-to-end performance (i.e., the Relative Cost Reduction and the Time Duration) of various heuristic-based candidate generation methods by replacing the original building block in each index advisor with three variants with different methods[2] and making other building blocks unchanged. (1) Permutation: columns are randomly permutated to generate indexes, which are further filtered out via the what-if calls of the database optimizer (i.e., reserve the ones that appear in the query plan); (2) SyntacticRule: the heuristic rules proposed in [29], which consider the syntactic characteristics of queries; (3) openGauss: the method utilized in [34, 64], which not only takes the syntactic query features into account but also the statistics of the indexed columns. On the other hand, we compare the performance of the learned

[2]We do not create variants with prefix-based expansion since it is tightly coupled with the underlying selection algorithm and is not applicable to all the index advisors.

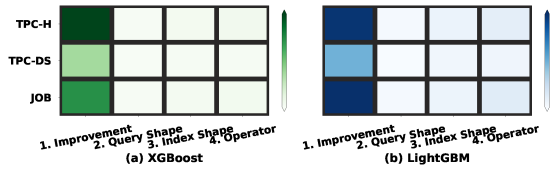

Figure 12: Distribution of Importance Scores for Four Input Signals in Learned Filter Model

Table 7: F1 Score of Different Filter Methods

Datasets	TPC-H			TPC-DS			JOB		
Method	$\delta > 0.05$	$\delta > 0.2$	$\delta > 0.5$	$\delta > 0.05$	$\delta > 0.2$	$\delta > 0.5$	$\delta > 0.05$	$\delta > 0.2$	$\delta > 0.5$
PostgreSQL [3]	0.94	0.85	0.82	0.90	0.85	0.63	0.93	0.81	0.67
openGauss [34]	**0.95**	0.89	0.86	0.89	0.87	0.67	**0.94**	0.84	0.75
DIS-XGBoost [49]	**0.95**	**0.91**	**0.88**	**0.94**	**0.90**	**0.84**	**0.94**	0.87	**0.81**
DIS-LightGBM [49]	**0.95**	**0.91**	0.87	0.93	**0.90**	0.83	**0.94**	0.87	0.73
DIS-RandomForest [49]	**0.95**	0.89	0.84	0.92	0.89	0.83	**0.94**	**0.88**	0.68

filter model with the methods leveraging the estimated statistics from the database optimizer. We implement multiple variants of the learned filter models in DISTILL [49] with four input signals: (1) Improvement: calculate and sum up the cost of each node in the query plan that the index can reduce; (2) Query Shape: represent each table by concatenating the type of the node in the query plan that involves the table with a bottom-up traversal; (3) Index Shape: replace each column of the index with the first operation (e.g., the selection or the join operation) in the query that utilizes the column; (4) Operator: represent each node in the query plan with the statistics of the involved indexed columns (e.g., the distinct ratio). We construct a dataset (split into the training/validation/testing set by 8:1:1) including the above features to train the models and compare their performance (i.e., the F1 Score of filtering out the useless index candidates) to identify the most critical input feature.

O10 (Generation Method): Candidates generated by rule-based construction can enhance the effectiveness of learning-based index advisors and the efficiency of heuristic-based index advisors. As shown in Figure 11, learning-based index advisors are more effective with candidates generated by rule-based construction (i.e., SyntacticRule and openGauss). For example, SWIRL obtains a higher Relative Cost Reduction, i.e., 21.04% with SyntacticRule and openGauss than 13.40% with Permutation on average. Besides, heuristic-based index advisors are more efficient with the rule-based construction method, especially over large datasets. For example, Relaxation requires 24.44× less time over TPC-DS benchmark and 2.92× over TPC-H benchmark with the rule-based construction method on average. It might be attributed to the candidates generated by rule-based construction are typically effective indexes with a small volume. Therefore, learning-based index advisors require less training effort to learn a good selection policy over these effective indexes, and heuristic-based index advisors spend less time overhead to explore a small set from the candidate space.

O11 (Learned Filter Model): Tree-based filter models effectively filter out useless index candidates, where the Improvement is the most critical input feature. As displayed in Table 7, tree-based filter models achieve a higher F1 Score than methods based on the estimated statistics from the database optimizers. For example, DIS-XGBoost achieves the F1 Score, i.e., 0.89 compared with the F1 Score, i.e., 0.82 of PostgreSQL over different filter thresholds δ on average. To further identify the most critical input feature that contributes to the effectiveness of these models, we display the distribution of the importance scores over the input features in Figure 12. We observe that Improvement is the most critical feature. It aligns with the functionality of indexes, which optimizes the operation of the node in the query plan (e.g., reduce the cost of Seq Scan by replacing it with Index Scan). Thus, models can estimate the impact of indexes and filter out candidates with a low impact.

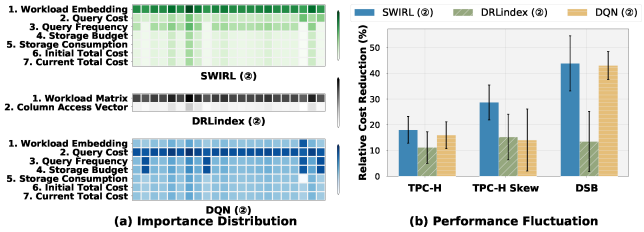

Figure 13: (a) Importance Distribution over State Representation and (b) Performance Variance over Same Workload

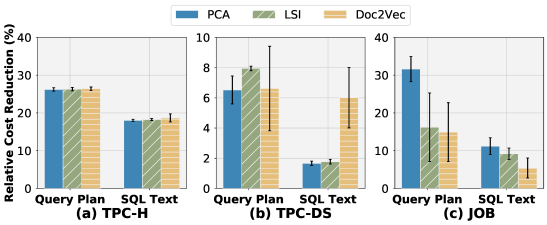

Figure 14: Performance over Various State Representation

5.2 Index Selection

Experimental Setting. Since heuristic-based index advisors adopt nearly the same greedy algorithms in Index Selection, we evaluate the performance of various learning-based index advisor variants. Specifically, we explore the influence of the state representation that captures the environmental information and distinguishes workloads in learning-based index advisors. We first calculate the importance scores of different features in the state representation by excluding them from the input (i.e., the corresponding feature values are set to zero) and observing the change of the Relative Cost Reduction. Then, we verify whether existing index advisors can return consistent indexes for the same workloads where queries are organized in different orders (i.e., the permutation invariance property [23]). Finally, we develop multiple index advisor variants with the workload representation based on (1) SQL Text or (2) Query Plan. These representations are further transformed into a fixed sized with three dimension reduction methods (i.e., PCA [52], LSI [21], and Doc2Vec [30]). The workloads are the same query templates with different parameter values and are split into the training/validation/testing set by 8:1:1.

O12 (State Representation): Learning-based index advisors are not stable in representing permutation invariance property, where query-plan-based workload representation is most critical. Figure 13 (a) displays the distribution of the importance scores of state representation features, where each row corresponds to a feature of the state, and each column is a workload sample. We observe the workload representation is the most critical feature in the state representation. Specifically, the Workload Embedding, the Workload Matrix, and the Query Cost are the most crucial features of SWIRL, DRLindex, and DQN, respectively. The intuition is that existing learning-based index advisors are workload-dependent and workload representation is crucial for them to identify effective indexes for the given workload during training. However, existing workload representations are problematic since they cannot preserve the permutation invariance property (i.e., return the same index for a permutated workload) [23]. As shown in Figure 14 (b), index advisors fail to yield consistent Relative Cost Reduction (i.e., return the same index) for the same workload where queries are randomly shuffled (i.e., organized in different orders). The underlying reason is that index advisors process queries in the workload in order (e.g., associate them with specific rows in a matrix), which contradicts the unordered nature of the workload processed in a batch manner. We also find that workload representations based on Query Plan are more effective than SQL Text. Figure 14 presents the performance of SWIRL with various workload representations. We notice that SWIRL obtains the respective Relative Cost Reduction, i.e., 18.08% and 10.03%, with workload representation based on Query Plan and SQL Text on average. Query Plan involves more index-specific information that reflects the optimization opportunity with indexes (e.g., building indexes with columns in Seq Scan can optimize it with Index Scan). Thus, workload representation based on Query Plan is more instructive in identifying effective indexes than on SQL Text.

5.3 Index Benefit Estimation

Experimental Setting. We evaluate the estimation accuracy and the end-to-end performance (i.e., lower query latency) of various benefit estimation models. (1) For the estimation accuracy, we assess the what-if calls of the database optimizers in a prevalent DBMS (i.e., PostgreSQL [3]) and an open-source DBMS (i.e., openGauss [34]). Besides, we evaluate the tree-based models (i.e., XGBoost and LightGBM [20]), the attention-based model (i.e., LIB [47]), and the transformer-based model (i.e., QueryFormer [60]). We also identify the critical input features of these models with explainable machine learning algorithms in [25]; (2) For the end-to-end performance, we integrate the learned models into the database and replace the database optimizer with them to calculate the index benefits during the selection process and the training process (for learning-based index advisors). We compare the query latency with the indexes returned by different index advisors using these learned models. The learned models are trained and tested based on the dataset (split into the training/validation/testing set by 8:1:1) containing the estimated information (i.e., the statistics in the query plan) and the actual statistics (i.e., the query latency).

O13 (Estimation Accuracy): Tree-based models with the input features based on the estimated cardinality and cost are more accurate than other estimation models. As displayed in Table 8, tree-based models outperform others across various datasets. Specifically, AI-XGBoost and AI-LightGBM obtain the 95th Q-Error, i.e., 18.26, which is 231.34× lower than the database optimizer and 171.50× lower than other learned models on average. It might be attributed to the well-defined index-related feature vectors tree models utilized to estimate the index benefit. However, other models require a training process with more effort (i.e., higher training difficulty) to effectively learn the correlations among input features. Figure 15 presents the importance distribution over the input features, where each row corresponds to the importance scores calculated by the explainable machine learning algorithms [25]. We observe that the estimated cardinality (i.e., Card), the estimated cost (i.e., Cost), and the predicate information (i.e., Predicate) are critical input features for these learned models. It aligns with the functionality of indexes, e.g., accelerating the retrieval over different selection predicates, especially the ones of high selectivity (calculated based on cardinality).

O14 (End-to-End Performance): Learned benefit estimation model with a higher accuracy might not contribute to better end-to-end performance. According to the results in Table 8 and

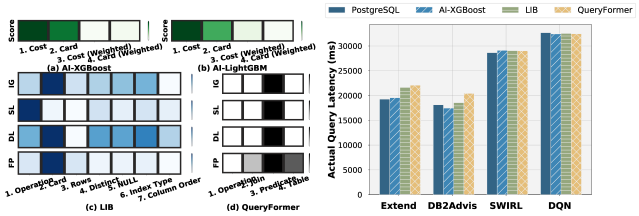

Figure 15: Importance Distribution over Input Feature and End-to-End Performance of Different Estimation Models

Table 8: Q-Error of Different Index Benefit Estimation Models

Datasets	TPC-H			TPC-DS			JOB		
Method	**Mean**	**90th**	**95th**	**Mean**	**90th**	**95th**	**Mean**	**90th**	**95th**
PostgreSQL [3]	21490.41	42.68	477.71	43003.55	1738.09	12445.39	762.33	46.35	555.13
openGauss [34]	11018.69	69.89	543.76	9348.81	2203.21	10943.32	272.05	53.94	494.00
AI-XGBoost [20]	88.91	**3.53**	**8.58**	358.31	8.06	36.56	**7.01**	**2.63**	**4.12**
AI-LightGBM [20]	**40.32**	4.78	12.40	**198.03**	8.85	42.96	9.00	2.91	4.96
LIB [47]	304.12	644.53	1272.73	7505.93	2835.82	14390.48	277.12	525.33	1160.78
QueryFormer [60]	256.67	4.63	13.58	348.20	**4.46**	**11.56**	593.37	868.11	2053.93

Figure 15, a more accurate learned model might not contribute to better end-to-end performance, i.e., return more effective indexes with a smaller query latency when integrated into the database to return index benefits during the selection process. For example, QueryFormer obtains a 95th Q-Error over TPC-H benchmark, i.e., 13.58 (34.18× lower than PostgreSQL), while the Query Latency is 32521.21*ms*, i.e., 1.58× than 20643.83*ms* of PostgreSQL on average. It might be attributed to the discrepancy between the optimization goal of learned benefit estimation models (i.e., higher estimation accuracy) and index selection (i.e., more effective indexes). Thus, achieving high accuracy in the estimation model might not contribute to a better indexing solution. Similar discoveries have been observed in prior studies of the learned cardinality model that a learned model with lower Q-Error (i.e., more accurate) might not contribute to a better query plan with lower latency [22].

6 SUMMARIZED FINDINGS

► **Application Scenario.** As shown in Figure 16, we summarize application scenarios of heuristic-based and learning-based index advisors. **(1)** ***Static Analytical Scenario:*** Heuristic-based index advisors can perform well over query-level selection with a small time overhead. Learning-based index advisors can identify more effective indexes than heuristic-based index advisors over workload-level selection, which includes diverse query patterns. Moreover, learning-based index advisors with offline learning policy are more efficient than heuristic-based index advisors and learning-based index advisors with online learning policy. However, learning-based index advisors exhibit higher performance variance and might encounter performance regression for a larger storage budget; **(2)** ***Dynamic Analytical Scenario:*** Learning-based index advisors can adapt to small workload drifts (e.g., varying query frequency) and medium workload drifts, e.g., query perturbations over datasets with small schema and uniform data distribution (e.g., TPC-H). Heuristic-based index advisors are more robust than learning-based index advisors over large workload drifts, e.g., query perturbations over complex datasets with complex schema and skewed data distribution (e.g., TPC-H Skew) and random workloads; **(3)** ***Transactional Scenario:*** Heuristic-based index advisors perform well over READ HEAVY workloads with simple transactional queries (i.e., a few filtering predicates). Moreover, they are more robust to data shifts than learning-based index advisors over WRITE HEAVY workloads.

► **Design Choice. (1)** ***Index Candidate Generation:*** Heuristic-based index advisors are more efficient, and learning-based index advisors are more effective with **Rule-based Construction** than **Random Column Permutation**, especially over large datasets. **Prefix-based Expansion** is highly dependent on the quality of the underlying selection algorithm and does not apply to all index advisors (e.g., methods with decremental iterative manner). **Learned Filter Model** of tree-based models are effective in filtering out useless index candidates, where the estimated index impact is the most critical input feature; **(2)** ***Index Selection:*** For **Heuristic Search Strategy**, the decremental iterative manner is more time-consuming than the incremental iterative manner, especially over large datasets with many candidates. For **Mathematical Solver**, it showcases constant efficiency over varying storage budgets. For **Learned Selection Policy**, workload representation based on the query plan is the most critical input feature; **(3)** ***Index Benefit Estimation:*** Tree-based models with the estimated cardinality and cost as the input feature are more accurate than other models.

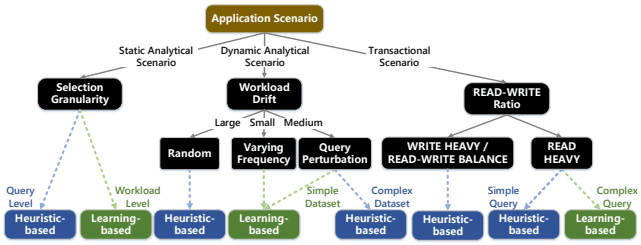

Figure 16: Application Scenario of Different Index Advisors

► **Future Direction.** (1) Further investigation of generalization is required for an advanced index advisor to handle various scenarios. Specifically, learning-based index advisors need to be more robust over query drifts and data shifts (e.g., with additional training processes of fast transfer learning techniques). Heuristic-based index advisors need to be more efficient over large-scale workloads (e.g., with effective index candidate filtering techniques); (2) More factors need to be addressed to facilitate the application of index advisors across real-world scenarios. For example, the consideration of the overhead for updating the indexes over the INSERT or the UPDATE statements; the modeling of essential and relevant database status (e.g., the existence of other physical structures and the currently available resources); the calculation of the index benefits with different index physical implementations (e.g., B-Tree or Hash).

7 CONCLUSION

We present the first comprehensive review of existing heuristic-based and learning-based index advisors, coupling this with extensive experiments to evaluate their performance and explore their suitable application scenarios. Additionally, we delve into the impact of the individual building blocks by constructing multiple index advisor variants, thereby identifying the most effective methods and the most influential factors. Finally, we summarize the findings and present guidance for users in selecting suitable index advisors for their application scenarios. We also offer an open-source testbed designed to ease the evaluations of various index advisors.

ACKNOWLEDGMENTS

Guoliang Li is the corresponding author. This work was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (62372390, 61925205, 62232009, 62102215), Zhongguancun Lab, Huawei, TAL education, and Beijing National Research Center for Information Science and Technology (BNRist).

REFERENCES

[1] HypoPG. *(Extension).* https://github.com/HypoPG/hypopg visited on 2024-06-12.

[2] Index Advisor (EA&B). *(Extended Version).* https://github.com/Beliefuture/Index_EAB visited on 2024-06-12.

[3] PostgreSQL. *(DBMS).* https://www.postgresql.org visited on 2024-06-12.

[4] TPC-DS Benchmark. *(TPC).* https://www.tpc.org/tpcds visited on 2024-06-12.

[5] TPC-H Benchmark. *(TPC).* https://www.tpc.org/tpch visited on 2024-06-12.

[6] Stefan Aulbach, Dean Jacobs, Alfons Kemper, and Michael Seibold. 2009. A comparison of flexible schemas for software as a service. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 881–888.

[7] Gabriel R. Bitran and A. G. Novaes. 1973. Linear Programming with a Fractional Objective Function. *Operation Research* 21, 1 (1973), 22–29.

[8] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transaction on Computational Intelligence and AI Games* 4, 1 (2012), 1–43.

[9] Nicolas Bruno and Surajit Chaudhuri. 2005. Automatic Physical Database Tuning: A Relaxation-based Approach. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 227–238.

[10] Alberto Caprara, Matteo Fischetti, and Dario Maio. 1995. Exact and Approximate Algorithms for the Index Selection Problem in Physical Database Design. *IEEE Trans. Knowl. Data Eng.* 7, 6 (1995), 955–967.

[11] S. Chaudhuri and V. Narasayya. 2020. Anytime Algorithm of Database Tuning Advisor for Microsoft SQL Server. https://www.microsoft.com/en-us/research/publication/. visited on 2024-06-12.

[12] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. 146–155.

[13] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*. 785–794.

[14] Michelangelo Conserva and Paulo E. Rauber. 2022. Hardness in Markov Decision Processes: Theory and Practice. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*. 14824–14838.

[15] 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 *Proceedings of the International Conference on Management of Data (SIGMOD)*. 666–679.

[16] Debabrata Dash, Neoklis Polyzotis, and Anastasia Ailamaki. 2011. CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads. *Proceedings of the International Conference on Very Large Databases (VLDB)* 4, 6 (2011), 362–372.

[17] Biplob K. Debnath, David J. Lilja, and Mohamed F. Mokbel. 2008. SARD: A statistical approach for ranking database tuning parameters. In *Proceedings of the International Conference on Data Engineering (ICDE Workshops)*. 11–18.

[18] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proceedings of the International Conference on Very Large Databases (VLDB)* 7, 4 (2013), 277–288.

[19] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek R. Narasayya. 2021. DSB: A Decision Support Benchmark for Workload-Driven and Traditional Database Systems. *Proceedings of the International Conference on Very Large Databases (VLDB)* 14, 13 (2021), 3376–3388.

[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 *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1241–1258.

[21] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.

[22] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proceedings of the International Conference on Very Large Databases (VLDB)* 15, 4 (2021), 752–765.

[23] Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. 2022. Learned Cardinality Estimation: An In-depth Study. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1214–1227.

[24] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *Proceedings of the International Conference on Innovative Data Systems Research (CIDR)*. 1–8.

[25] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. 2020. Captum: A unified and generic model interpretability library for PyTorch. *arXiv Preprint* (2020). https://arxiv.org/abs/2009.07896

[26] 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. *Proceedings of the International Conference on Very Large Databases (VLDB)* 13, 11 (2020), 2382–2395.

[27] Jan Kossmann, Alexander Kastius, and Rainer Schlosser. 2022. SWIRL: Selection of Workload-aware Indexes using Reinforcement Learning. In *Proceedings of the 25th International Conference on Extending Database Technology (EDBT)*. 2:155–2:168.

[28] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 489–504.

[29] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2020. An Index Advisor Using Deep Reinforcement Learning. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*. 2105–2108.

[30] Quoc V. Le and Tomás Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the International Conference on Machine Learning (ICML) (JMLR Workshop and Conference Proceedings)*, Vol. 32. 1188–1196.

[31] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[32] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proceedings of the International Conference on Very Large Databases (VLDB)* 9, 3 (2015), 204–215.

[33] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1920–1933.

[34] Guoliang Li, Xuanhe Zhou, Ji Sun, Xiang Yu, Yue Han, Lianyuan Jin, Wenbo Li, Tianqing Wang, and Shifu Li. 2021. openGauss: An Autonomous Database System. *Proceedings of the International Conference on Very Large Databases (VLDB)* 14, 12 (2021), 3028–3041.

[35] Gabriel Paludo Licks, Júlia Mara Colleoni Couto, Priscilla de Fátima Miehe, Renata De Paris, Duncan Dubugras A. Ruiz, and Felipe Meneguzzi. 2020. SmartIX: A database indexing agent based on reinforcement learning. *Applied Intelligence* 50, 8 (2020), 2575–2588.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[37] Vivek R. Narasayya and Surajit Chaudhuri. 2022. Multi-Tenant Cloud Data Services: State-of-the-Art, Challenges and Opportunities. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 2465–2473.

[38] Parimarjan Negi, Ziniu Wu, Andreas Kipf, Nesime Tatbul, Ryan Marcus, Sam Madden, Tim Kraska, and Mohammad Alizadeh. 2023. Robust Query Driven Cardinality Estimation under Changing Workloads. *Proceedings of the International Conference on Very Large Databases (VLDB)* 16, 6 (2023), 1520–1533.

[39] R. Malinga Perera, Bastian Oetomo, Benjamin I. P. Rubinstein, and Renata Borovica-Gajic. 2021. DBA bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 600–611.

[40] Zahra Sadri, Le Gruenwald, and Eleazar Leal. 2020. DRLindex: deep reinforcement learning index advisor for a cluster database. In *Proceedings of the Symposium on International Database Engineering and Applications (IDEAS)*. 11:1–11:8.

[41] Zahra Sadri, Le Gruenwald, and Eleazar Leal. 2020. Online Index Selection Using Deep Reinforcement Learning for a Cluster Database. In *Proceedings of the International Conference on Data Engineering (ICDE Workshops)*. 158–161.

[42] Rainer Schlosser and Stefan Halfpap. 2020. A Decomposition Approach for Risk-Averse Index Selection. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*. 16:1–16:4.

[43] Rainer Schlosser, Jan Kossmann, and Martin Boissier. 2019. Efficient Scalable Multi-attribute Index Selection Using Recursive Strategies. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 1238–1249.

[44] Rainer Schlosser, Marcel Weisgut, Leonardo Hübscher, and Oliver Nordemann. 2023. Robust Index Selection for Stochastic Dynamic Workloads. *SN Computer Science* 4, 1 (2023), 59.

[45] Karl Schnaitter and Neoklis Polyzotis. 2012. Semi-Automatic Index Tuning: Keeping DBAs in the Loop. *Proceedings of the International Conference on Very Large Databases (VLDB)* 5, 5 (2012), 478–489.

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

[47] Jiachen Shi, Gao Cong, and Xiaoli Li. 2022. Learned Index Benefits: Machine Learning Based Index Performance Estimation. *Proceedings of the International Conference on Very Large Databases (VLDB)* 15, 13 (2022), 3950–3962.

[48] 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 *Proceedings of the International Conference on Management of Data (SIGMOD)*. 660–673.

[49] 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. *Proceedings of the International Conference on Very Large Databases (VLDB)* 15, 10 (2022), 2019–2031.

[50] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *Proceedings of the International Conference on Very Large Databases (VLDB)* 13, 3 (2019), 307–319.

[51] Ji Sun, Jintao Zhang, Zhaoyan Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation: A Design Space Exploration and A Comparative Evaluation. *Proceedings of the International Conference on Very Large Databases (VLDB)* 15, 1 (2021), 85–97.

[52] Michael E. Tipping and Christopher M. Bishop. 1999. Mixtures of Probabilistic Principal Component Analysers. *Neural Computation* 11, 2 (1999), 443–482.

[53] 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 *Proceedings of the International Conference on Data Engineering (ICDE)*. 101–110.

[54] Jiaqi Wang, Tianyi Li, Anni Wang, Xiaoze Liu, Lu Chen, Jie Chen, Jianye Liu, Junyang Wu, Feifei Li, and Yunjun Gao. 2023. Real-time Workload Pattern Analysis for Large-scale Cloud Databases. *arXiv Preprint* (2023). https://arxiv.org/abs/2307.02626

[55] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *Proceedings of the International Conference on Very Large Databases (VLDB)* 14, 9 (2021), 1640–1654.

[56] Kyu-Young Whang. 1987. Index Selection in Relational Databases. *Foundations of Data Organization* (1987), 487–500.

[57] 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 *Proceedings of the International Conference on Data Engineering (ICDE)*. 1081–1092.

[58] 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 *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1528–1541.

[59] Lixi Zhang, Chengliang Chai, Xuanhe Zhou, and Guoliang Li. 2022. Learned-SQLGen: Constraint-aware SQL Generation using Reinforcement Learning. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 945–958.

[60] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. QueryFormer: A Tree Transformer Model for Query Plan Representation. *Proceedings of the International Conference on Very Large Databases (VLDB)* 15, 8 (2022), 1658–1670.

[61] 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 International Conference on Information and Knowledge Management (CIKM)*. 5133–5137.

[62] Wei Zhou, Chen Lin, Xuanhe Zhou, Guoliang Li, and Tianqing Wang. 2024. TRAP: Tailored Robustness Assessement for Index Advisors via Adversarial Perturbation. In *Proceedings of the International Conference on Data Engineering (ICDE)*. to appear.

[63] Xuanhe Zhou, Cheng Chen, Kunyi Li, Bingsheng He, Mian Lu, Qiaosheng Liu, Wei Huang, Guoliang Li, Zhao Zheng, and Yuqiang Chen. 2023. FEBench: A Benchmark for Real-Time Relational Data Feature Extraction. *Proceedings of the International Conference on Very Large Databases (VLDB)* 16, 12 (2023), 3597–3609.

[64] 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 *Proceedings of the International Conference on Data Engineering (ICDE)*. 2196–2208.

[65] Yiwen Zhu, Yuanyuan Tian, Joyce Cahoon, Subru Krishnan, Ankita Agarwal, Rana Alotaibi, Jesús Camacho-Rodríguez, Bibin Chundatt, Andrew Chung, Niharika Dutta, Andrew Fogarty, Anja Gruenheid, Brandon Haynes, Matteo Interlandi, Minu Iyer, Nick Jurgens, Sumeet Khushalani, Brian Kroth, Manoj Kumar, Jyoti Leeka, Sergiy Matusevych, Minni Mittal, Andreas Müller, Kartheek Muthyala, Harsha Nagulapalli, Yoonjae Park, Hiren Patel, Anna Pavlenko, Olga Poppe, Santhosh Ravindran, Karla Saur, Rathijit Sen, Steve Suh, Arijit Tarafdar, Kunal Waghray, Demin Wang, Carlo Curino, and Raghu Ramakrishnan. 2023. Towards Building Autonomous Data Services on Azure. In *Proceedings of the International Conference on Management of Data (SIGMOD Companion)*. 217–224.