TRAP: Tailored Robustness Assessment for Index Advisors via Adversarial Perturbation

***Abstract*—Recently, many index advisors are proposed to automatically build indexes that can improve query performance. However, they mainly consider performance improvement in static scenarios. Their *robustness*, i.e., stable performance in dynamic scenarios (e.g., with minor workload changes), has not been well investigated. In this paper, we address the challenges of assessing the index advisor's robustness from the following aspects. First, we introduce perturbation-based workloads for robustness assessment and identify three typical perturbation constraints that occur in real scenarios. Second, with the perturbation constraints, we formulate the generation of perturbed queries as a sequence-to-sequence problem and propose `TRAP` (Tailored Robustness assessment via Adversarial Perturbation) to pinpoint the performance loopholes of index advisors. Third, to generalize to various index advisors, we place `TRAP` in an opaque-box setting (i.e., with little knowledge of the index advisors' internal design), and we propose a two-phase training paradigm to efficiently train `TRAP` without the need of elaborate annotated data. Fourth, we conduct comprehensive robustness assessments for ten existing index advisors on both standard benchmarks and real workloads. Our findings reveal that these index advisors are vulnerable to the workloads generated by `TRAP`. Finally, based on the assessment results, we shed light on insights to enhance the robustness of different index advisors. For example, learning-based index advisors can benefit from adopting a fine-grained state representation and a candidate pruning strategy.**

I. Introduction

Indexes are crucial in database optimization [1]. Traditionally, indexes are built by expert database administrators (DBAs) [2] who analyze the workload characteristics and create indexes that are likely to accelerate workload execution. Obviously, this procedure is labor-intensive. To reduce the manual effort and automate the selection process, index advisors have been extensively studied [3, 4, 5, 6, 7, 8, 9, 10].

Early index advisors are mostly heuristic-based [3, 4, 5, 6, 11, 12], where indexes that maximize predefined criteria (e.g., the relative cost reduction) are greedily added or removed. However, they have two limitations. First, they cannot well capture the correlations between the query patterns and data distributions. Second, they use heuristics to select indexes from a large set of candidates and are often stuck in a sub-optimal solution. To address these limitations, learning-based index advisors [7, 8, 9, 10, 13, 14] have been proposed. They capture the syntactic query patterns associated with the index (e.g., columns in the predicates) based on the training workloads. They mostly adopt a reinforcement learning framework and take actions to select indexes based on the current state (e.g., the workload characteristics).

Although existing index advisors attempt to achieve high accuracy in selecting appropriate indexes to reduce the cost of a static workload [2], they neglect an important factor – *robustness*. The robustness of an index advisor is whether it can adapt to dynamic workloads and maintain stable performance without expensive model re-training [15, 16]. The robustness of existing index advisors has not been fully investigated. Overall, when assessing the robustness of an index advisor, a pivotal consideration lies in devising appropriate testing workloads that satisfy two critical criteria. (1) *Real-world Relevance*: the workloads should reflect real-world production systems with a high probability of occurrence; (2) *Efficacy in Loophole Detection*: the workloads must pinpoint the performance loopholes of the index advisors being assessed. These two criteria are of paramount importance in measuring the index advisor's robustness.

To reflect the robustness of index advisors in practice, the testing workloads should mimic workload drifts [17, 18, 19, 20] resulting from query changes in daily life, i.e., to generate workloads that might occur in real-world scenarios. Our key observation is that most queries in real production systems and open-source benchmarks are *perturbed* variants of a small set of templates. These templates undergo changes due to shifts in user behavior and business demands [21, 22]. As shown in Figure 1, 1.7 billion queries executed in the industry (e.g., the Fortune 500 and the Global 2000 companies) are based on 31 million query templates with different parameter bindings [23] (Figure 1(a)); the queries of eight open-source benchmarks [19, 24, 25, 26, 27] are generated from a small number of well-crafted templates (Figure 1(b)).

Furthermore, perturbations lead to "variants" of the "original" workloads that the index advisors are "supposed to be competent" to select appropriate indexes, both from an algorithm point of view (e.g., a learning-based index advisor is well-trained on similar workloads [28]) and from an application point of view (e.g., index advisors are required to adapt to minor workload changes in practice). Therefore, we propose to adopt queries generated based on perturbations over the original workloads and conduct a robustness assessment based on these workloads.

To comprehensively assess index advisors' robustness, the testing workloads should be *tailored* to reveal the performance loopholes of each index advisor. However, it presents the following challenges when designing a framework to generate such workloads. First, because of the wide-ranging diversity of SQL queries in terms of literal patterns (e.g., various database schemas and multiple clauses), a great number of operations can be performed to perturb the SQL (e.g., adding a specific column in the SELECT clause). Defining a distinct action rule

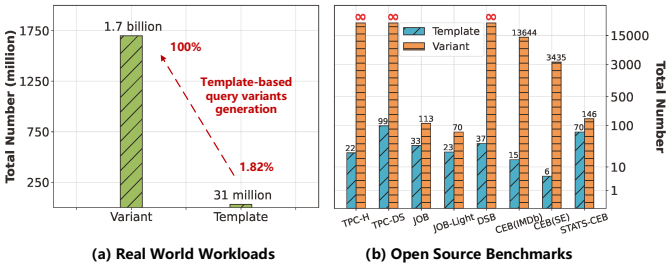

Fig. 1: Most queries in real-world workloads (a) and open-source benchmarks (b) are variants perturbed from a limited number of templates (∞ denotes the number is unlimited).

for each operation becomes impractical due to the excessive action space (**C1**). Second, given the significant disparities in the internal architecture of index advisors (e.g., the adopted strategies and learning paradigms), it becomes imperative to design a unified generation framework, which involves minimal knowledge and generalizes to various index advisors (**C2**). Third, the generation process should be efficient, i.e, produce executable queries and reduce the number of invalid queries (e.g., violate the given perturbation constraints), precisely target the performance loopholes of the index advisors (**C3**).

To address **C1**, we formulate perturbation over queries in the original workloads as a sequence-to-sequence problem. Given the input SQL, the perturbed query is synthesized by generating each SQL token based on an encoder-decoder network. Therefore, all the operations (e.g., add or replace a specific token) can be implemented in a unified manner, and the size of the action space is independent of the number of operations. To address **C2**, we propose a tailored workload generation framework `TRAP` based on a two-phase training paradigm. It generates a workload that intends to degrade the performance of the index advisors by exploring perturbation combinations via reinforcement learning without any prior knowledge of their internal designs. Moreover, to enhance the sample efficiency in reinforcement learning, `TRAP` performs index advisor independent pretraining, effectively initializing the agent with an understanding of the SQL semantics. To address **C3**, we design a novel tree-based structure to further restrict the action space of the permissible SQL tokens. This structure ensures that each generated query satisfies the perturbation constraints and strictly adheres to the SQL grammar. Furthermore, a reward function based on the learned index utility is adopted to provide more accurate rewards associated with the performance drops of index advisors.

To the best of our knowledge, this paper is the first attempt to thoroughly study the robustness of index advisors. The main contributions of this paper are summarized below.

- We introduce the concept of perturbation-based adversarial workloads based on the observations from typical workload drifts to measure the robustness of index advisors (Section III).
- We formulate the generation of the perturbed queries as a sequence-to-sequence problem and propose a generation framework `TRAP` based on an encoder-decoder architecture to implement multiple perturbations in a unified manner (Section IV-A).
- We adopt a two-phase training paradigm to generate effective adversarial workloads over various index advisors with little knowledge about their internal designs (Section IV-B / IV-C).
- We design a novel structure to ensure the validity of the perturbed SQL and utilize a learned index utility model to provide more accurate feedback (Section IV-D).
- We conduct a thorough robustness assessment on ten existing index advisors over both open-source benchmarks and real-world datasets (Section V).
- We reveal insightful findings and discoveries from the assessments, which can facilitate the design of more robust index advisors (Section VI).

II. Related Work

A. Index Advisor

Heuristic-based index advisors *incrementally* add [3, 4, 5] or *decrementally* [6, 29] remove candidate indexes (i.e., single-column indexes [6] or multi-column indexes [3, 4, 5, 29]) according to the predefined criteria (e.g., the relative cost reduction [5, 6, 11, 12] or the benefit-per-storage ratio [3, 4, 29]).
Learning-based index advisors apply machine learning techniques, primarily based on Reinforcement Learning (RL) [30], where an *agent* (typically a neural network [31]) outputs an *action* (i.e., the selected indexes) based on the current *state* to maximize a *reward* function (e.g., the estimated cost reduction [8, 9, 10, 13, 32], the benefit-per-storage ratio [7] or the actual runtime [1, 14]). There are various choices of the state representations to incorporate features of the workloads at different levels (more details in Section VI-A).
Difference with our work. Most of the index advisors, especially the learning-based ones, are only assessed on static workloads. Although a few recent attempts have considered dynamic workloads [1, 7], the assessments are conducted on the testing workloads split from a predefined set of workloads, which only contain limited query variants and fails to reflect the typical workload drifts. Therefore, the robustness of both heuristic-based and learning-based index advisors over workload drifts has never been thoroughly assessed.

B. SQL Generation

Heuristic-based SQL generation methods use various rules. For example, SQLsmith [33] randomly synthesizes queries by walking through the parse tree. TLP [34] derives multiple queries by partitioning the results from the original query. Recently, learning-based SQL generation methods adopt Generative Adversarial Network (GAN) [35] or the reinforcement learning framework [36]. Instead of generating queries from scratch, some studies perform a series of rewrite transformations to optimize the original query. Sia [37] replaces the predicates in a query with valid but weaker ones learned by a classifier over the columns. LearnedRewrite [38] adopts the Monte Carlo Tree Search (MCTS) algorithm to find a near-optimal rewrite order from a set of query rewrite rules.

Difference with our work. First, our work aims to generate perturbed queries due to typical workload drifts. Existing studies, such as heuristic-based methods or generating from scratch, will produce many queries that might never occur in the real world. Second, the generation process is instructed to intentionally degrade the index advisor's performance. Existing methods need a large volume of queries to find the performance loopholes due to the mismatch among the generation goals (refer to the results in Section V-B).

C. Adversarial Attack

Adversarial attack has received much attention in the Computer Vision (CV) and Natural Language Processing (NLP) fields. There are two categories of adversarial attack: (1) the evasion attack [39], where a testing sample similar to the original input sample is constructed to deteriorate the effectiveness of a well-trained or deployed model, and (2) the data poisoning attack [40], where samples are injected into the training set to mislead the model's training procedure.

Difference with our work. Recently, data poisoning attack has been explored on learned index structure [41]. Our work belongs to evasion attack that aims to assess the robustness of index advisors. The opaque-box setting (i.e., the design details are not exposed) requires the assessment procedure to be less intrusive and more generalized. The similarity constraint adopted in this work is different from existing evasion attacks [39], i.e., the proposed perturbations over the SQL queries reflect the typical workload drifts so that the robustness of index advisors can be more accurately and practically measured.

III. Problem Definition

DEFINITION 3.1 (Index Advisor): Given a dataset $\mathbf{d}$, a workload $\mathcal{W}$ which contains a set of queries and the associated weights (e.g., the frequency of query $\mathbf{q}$ is $\mathbf{e}$), i.e., $\mathcal{W} = \{(\mathbf{q}, \mathbf{e})\}, |\mathcal{W}| \geq 1$, an index advisor f returns a set of indexes $\mathcal{I} = f(\mathcal{W}, \mathbf{d})$ based on its own internal mechanism.

DEFINITION 3.2 (Index Utility): The index advisor f's utility $u(\mathcal{W}, \mathbf{d}, f)$ for a workload $\mathcal{W}$ on a dataset $\mathbf{d}$ is the relative cost reduction with the selected indexes $\mathcal{I} = f(\mathcal{W}, \mathbf{d})$, compared with a baseline index configuration $\mathcal{I}^b$,

$$u(\mathcal{W}, \mathbf{d}, f) = 1 - \frac{c(\mathcal{W}, \mathbf{d}, \mathcal{I})}{c(\mathcal{W}, \mathbf{d}, \mathcal{I}^b)} \tag{1}$$

where $c(\mathcal{W}, \mathbf{d}, \mathcal{I})$ is a cost metric of running workload $\mathcal{W}$ on dataset $\mathbf{d}$ given the selected indexes $\mathcal{I}$.

The index utility u measures the optimization ability of the indexes based on the workload cost, similar to the primary focus of previous studies [1, 2, 7, 8, 9]. The baseline $\mathcal{I}^b$ allows more flexibility in measuring the index's quality. For example, if the baseline $\mathcal{I}^b$ is the default indexes, we require the index advisor to select indexes that outperform the default indexes (i.e., $u > 0$) [42] (More details about $\mathcal{I}^b$ are in Section V).

In real production systems and open-source benchmarks, workloads are rarely static [21, 22, 23]. The robustness of an index advisor over dynamic workloads can be measured by the fluctuation of index utility on different workloads. Next, we define the robustness of an index advisor.

DEFINITION 3.3 (Index Advisor's Robustness): Given a dataset $\mathbf{d}$, an original workload $\mathcal{W}$ (e.g., the current workload or the training workload for a learning-based index advisor), another workload $\mathcal{W}'$ (e.g., generated by the robustness assessment process for performance comparison with $\mathcal{W}$), and an index advisor f with $u(\mathcal{W}, \mathbf{d}, f) > \theta$ ($\theta \geq 0$), the robustness is defined as the Index Utility Decrease Ratio (*IUDR*) on $\mathcal{W}'$,

$$IUDR = 1 - \frac{u(\mathcal{W}', \mathbf{d}, f)}{u(\mathcal{W}, \mathbf{d}, f)} \tag{2}$$

As introduced in Section I, robustness is the ability of index advisors to provide high and stable performance over dynamic workloads without updating the index advisor. Thus, Definition 3.3 requires the index advisor f to be *properly operating* on the original workload $\mathcal{W}$ with $u(\mathcal{W}, \mathbf{d}, f) > \theta$, where the threshold θ is user-defined. A larger value of θ indicates the index advisor should provide higher performance. Note that we also require the generated workload $\mathcal{W}'$ to be sargable [43, 44] (i.e., can be optimized by a set of indexes). Specifically, we summarize the categories of query changes that make a query non-sargable (more details in Section VI-C). Then these non-sargable workloads are detected, verified, and filtered out from the robustness assessment process. According to Equation 2, a smaller *IUDR* indicates a smaller performance gap between $\mathcal{W}$ and $\mathcal{W}'$, i.e., more stable and robust performance.

Intuitively, $\mathcal{W}'$ must (1) *simulate workloads under workload drifts* to more accurately measure robustness in practice and (2) *target the weakness of index advisors* so that the performance gap between $\mathcal{W}$ and $\mathcal{W}'$ can detect whether the performance of the index advisor is truly stable. These two assumptions motivate us to give the following definition.

DEFINITION 3.4 (Perturbation-based Adversarial Workload): Given an original workload $\mathcal{W}$, a dataset $\mathbf{d}$, a perturbation-based adversarial workload $\mathcal{W}'$ is generated to assess the robustness of an index advisor f, $\mathcal{W}' = \{(\mathbf{q}', \mathbf{e}')\}$ consists of a set of queries and their weights (e.g., the occurring frequency). Each unseen query (i.e., a query that has not been included in the original workload) is generated by adding slight perturbations with an adversarial intent to the original workload, i.e., $\forall \mathbf{q}' \in \mathcal{W}' \setminus \mathcal{W}, |\mathbf{q}'| \geq 1, \exists \mathbf{q} \in \mathcal{W}, k(\mathbf{q}, \mathbf{q}') < \epsilon, \mathbf{e}' = \mathbf{e}, s.t., u(\mathcal{W}', \mathbf{d}, f) < u(\mathcal{W}, \mathbf{d}, f)$, where $k(\cdot, \cdot)$ is a distance metric, e.g., the edit distance.

We now explain the key concepts in Definition 3.4.

- **Perturbation-based Workload.** Definition 3.4 uses *perturbed queries* to simulate common query changes under workload drifts. Perturbed queries are widely observed in open-source benchmarks [24, 26, 27, 45, 46, 47, 48], where queries are variants of templates with different (1) query payloads (e.g., columns in the SELECT clause), (2) parameter values, and (3) filter predicates. Perturbation can be considered a meaningful template augmentation, e.g., additional predicates correspond to more fine-grained data slicing over the tuples without changing the original queries' semantics greatly [26]. Thus, for the purpose of robustness

assessment, it is better to use the perturbed queries instead of generating queries from scratch because the latter leads to many meaningless queries that never occur in real scenarios. Furthermore, perturbed queries are more appropriate for assessing the learning-based index advisors. By restricting a small perturbation, we generate workloads that a well-trained index advisor is "supposed" to perform well but fails to do so, exposing the vulnerability of the learning-based index advisors. On the contrary, generating from scratch might incur an inevitable performance drop for learning-based index advisors since workloads contain irrelevant query patterns to the original ones [17, 18, 19, 20].

- **Distance Metric.** Definition 3.4 uses SQL-level differences (i.e., edit distance) to quantify the degree of perturbation because changes in user behavior and business demands can be uniformly implemented by a series of edit operations. We do not use plan-level difference because the transformation from a SQL to a query plan is more pertinent to the optimizer [47] and is more appropriate to assess their robustness. The user-defined parameter ϵ controls the degree of the perturbation and indicates the maximal number of tokens that can be changed. For example, if $\epsilon = 5$, queries changed within five tokens are considered to be slight perturbations without strict amplitude ordering.
- **Adversarial Intent.** Definition 3.4 uses $u(\mathcal{W}', \mathbf{d}, f) < u(\mathcal{W}, \mathbf{d}, f)$ to indicate that $\mathcal{W}'$ is generated with an adversarial intent and targets the weakness of the given index advisor, i.e., leading to their performance drops.

Considering the common query changes due to typical workload drifts [17, 18, 19, 20], we present three types of perturbation constraints. These perturbation constraints differ in the types of tokens that can be modified (Table I)[1].

- **`Value Only Perturbation`** [18, 20] reflects the most common template-based workload drifts where queries are variants of the same set of pre-defined templates (e.g., TPC-H, TPC-DS, DSB [26]). For example, an online retailer may issue a series of queries to compare the sales figures of the same product in different seasons, pricing strategies, or marketing campaigns. This perturbation constraint only allows modifications on the predicate values, which correspond to the templates with placeholders. An example is displayed in Table I, the value in the predicate `t.kind_id = 1` is replaced with `3` to retrieve a different kind of movie.
- **`Column Consistent Perturbation`** [19, 27] mimics the workload drift when users operate on the same set of columns in daily transactions (e.g., CEB [19], STATS [27]). For example, a customer changes the order of columns in the search results of an E-commerce website to display products according to different preferences. This perturbation constraint allows modifications on columns and values, but the modified columns can only be chosen from the original column set. As shown in Table I, the arrangement of columns in the ORDER BY clause is changed to obtain results in different time order, i.e., "`t.production_year, t.series_years`" → "`t.series_years, t.production_year`".
- **`Shared Table Perturbation`** [17, 19, 24, 25] simulates the scenario when users in an exploratory analysis change the payloads or add new predicates to the original queries (e.g., JOB [24], CEB [19]). For instance, a sales analyst may execute analysis workloads with different filter predicates for the date range, product category, or customer demographics to uncover trends or patterns and assist inventory management decisions. The `Shared Table Perturbation` restricts the perturbed queries $\mathbf{q}'$ to be operated on the same table schema as $\mathbf{q}$, and allows modifications on other token types. An example is shown in Table I, where the query variant contains a new payload (`ci.note`) and a new predicate (`n.gender = 'f'`) to obtain additional movie cast information of the actress.

TABLE I: Legal token types that can be modified (colored in black) under different constraints. The original query is a simplified SQL in JOB [24].

Original Query q	SELECT t.title, n.name FROM title AS t, cast_info AS ci, name AS n WHERE t.id = ci.movie_id AND ci.person_id = n.id AND t.kind_id = 1 ORDER BY t.production_year, t.series_years				
Perturbation	**Column**	**Value**	**Conjunction**	**Operator**	**Aggregator**
Value Only	-	✓	-	-	-
Example Query q′ for Value Only	SELECT t.title, n.name FROM title AS t, cast_info AS ci, name AS n WHERE t.id = ci.movie_id AND ci.person_id = n.id AND t.kind_id = ***3*** ORDER BY t.production_year, t.series_years				
Column Consistent	✓	✓	-	-	-
Example Query q′ for Column Consistent	SELECT t.title, n.name FROM title AS t, cast_info AS ci, name AS n WHERE t.id = ci.movie_id AND ci.person_id = n.id AND t.kind_id = 1 ORDER BY ***t.series_years, t.production_year***				
Shared Table	✓	✓	✓	✓	✓
Example Query q′ for Shared Table	SELECT t.title, n.name, ***ci.note*** FROM title AS t, cast_info AS ci, name AS n WHERE t.id = ci.movie_id AND ci.person_id = n.id AND t.kind_id = ***3 AND n.gender = 'f'*** ORDER BY t.production_year, t.series_years				

IV. Adversarial Workload Generation

In this section, we introduce the details of `TRAP`. As shown in Figure 2, `TRAP` contains several modules to generate effective adversarial workloads. First, to cope with the difficulty of integrating various perturbations in a unified manner, `TRAP` formulates the generation of perturbed queries as a sequence-to-sequence problem based on an *Encoder Decoder Network*. Nevertheless, sequence models have shown catastrophic forgetting issues that lead to the information loss of previous input. To address this problem, `TRAP` adopts a SQL context attention mechanism to capture the characters of SQL (e.g., the overall query structure) (Section IV-A). Second, to tailor adversarial workloads for each index advisor, `TRAP` adopts a two-phase training paradigm. Given the limited knowledge and requirement of the generalization ability to index advisors with diverse internal designs, `TRAP` resorts to *Reinforced Perturbation Policy Learning*, where the encoder-decoder model acts as the agent. However, reinforcement learning typically suffers from time-consuming training procedures due to the sample inefficiency problem in obtaining a good policy. To enhance the training efficiency, `TRAP` employs an *Index Advisor Independent Pretraining*, where the agent is bootstrapped with SQL-related knowledge (e.g., the semantics of the queries). This phase is an offline one-time effort, independent of any index advisor, and transfers the knowledge to RL. Moreover, since the reward might be inaccurate as it is calculated by

[1]Note that the join predicates over columns, i.e., the join graph are not allowed to be modified considering the semantic meaningfulness of the query.

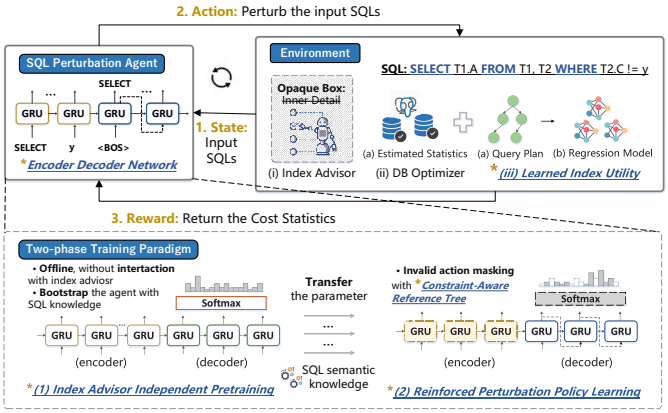

Fig. 2: Overview of the Adversarial Workload Generation Framework TRAP.

the estimated statistics from the optimizer, a *Learned Index Utility* model is adopted. This model enables more accurate quantification of performance drops of index advisors and provides more explicit guidance for TRAP to effectively target the performance loopholes (Section IV-B / IV-C). Finally, a vanilla sequence model fails to guarantee the output queries meet the perturbation constraints. Therefore, a novel structure, i.e., *Constraint-Aware Reference Tree* is integrated into the agent. This structure strictly restricts the permissible tokens at each step and enables TRAP to accommodate various perturbations constraints seamlessly (Section IV-D).

A. Perturbation via Encoder-Decoder Network

First, we explain how to generate the perturbed queries based on **the encoder-decoder network equipped with a SQL context attention mechanism** and implement various perturbations in a unified manner. As shown in Figure 3, the encoder of TRAP is a Bi-directional Gated Recurrent Unit (Bi-GRU) [49] layer, which consists of a forward GRU unit GRU^f and a backward GRU unit GRU^b. The decoder is another GRU layer. We leverage GRU since it is lightweight and effective compared with the transformer-based models [50] (refer to the results in Section V-C). For a SQL query[2] of n tokens $\mathbf{q} =< \mathbf{q}_1, \dots, \mathbf{q}_n >$, TRAP proceeds as follows.

Step 1: The encoder reads the query sequentially and returns a hidden vector for the token at position i, $\mathbf{h}_i = [[\mathbf{h}_i^f], [\mathbf{h}_i^b]] (i = 1, \dots, n)$, where $\mathbf{h}_i^f = GRU^f(\mathbf{q}_i, \mathbf{h}_{i-1}^f)$, $\mathbf{h}_i^b = GRU^b(\mathbf{q}_i, \mathbf{h}_{i+1}^b)$ and $[,]$ is the concatenation operation.

Step 2: Suppose the output query $\mathbf{q}' =< \mathbf{q}'_1, \dots, \mathbf{q}'_m >$ contains m tokens, the decoder outputs $\mathbf{q}'$ in a token-by-token manner. At each step t $(t = 1, \dots, m)$, the decoder's GRU cell returns a hidden vector $\mathbf{s}_t = GRU(\mathbf{q}'_{t-1}, \mathbf{s}_{t-1})$ based on the previously generated tokens.

Step 3: A context vector $\mathbf{c}_t$ is computed to encapsulate the significant information of the whole input SQL query. Since sequence models have shown catastrophic forgetting issues [51] in deriving $\mathbf{s}_t$, we leverage the attention mechanism [52] to learn to attend to different parts of the SQL.

[2]Note that we illustrate with a single SQL query. However, our framework can support multi-query workloads by concatenating the queries.

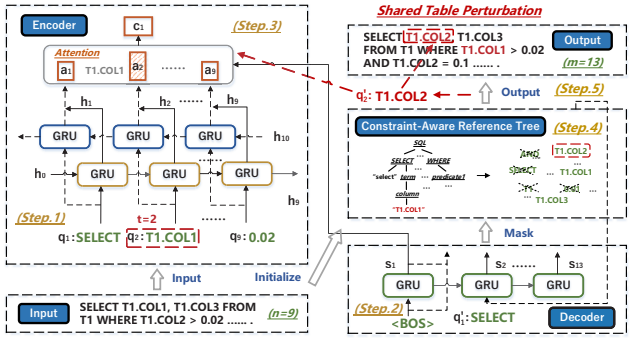

Fig. 3: Illustration of Encoder-Decoder Network in TRAP.

$$
\begin{aligned}
e_i^t &= \mathbf{v}^T tanh(\mathbf{W_h h_i} + \mathbf{W_s s_t} + \mathbf{b}), (i = 1, ..., n), \\
a_i^t &= \frac{\exp(e_i^t)}{\sum_{i=1}^{n} \exp(e_j^t)}, (j = 1, ..., n), \\
\mathbf{c_t} &= \sum_i a_i^t \mathbf{h_i}
\end{aligned} \tag{3}
$$

where $\mathbf{h}_i$ is the encoder's hidden state for token i, $\mathbf{s}_t$ is the decoder's hidden at timestep t, e_i^t computes the matching score between $\mathbf{h}_i$ and $\mathbf{s}_t$, a_i^t is the attention weight that normalizes the matching score $\mathbf{e}^t$. The context vector $\mathbf{c_t}$ aggregates over all the encoder's hidden states with the attention weights. $\mathbf{v}, \mathbf{W}, \mathbf{b}$ are learnable parameters.

Step 4: The decoder forms a legitimate vocabulary $\mathcal{V}^t$ by masking invalid tokens to guarantee the validity of the output query based on the `Constraint-Aware Reference Tree` (more details in Section IV-D).

Step 5: Given the context vector $\mathbf{c}_t$, previously generated output $\mathbf{q}'_{<t}$, and the input query $\mathbf{q}$, the decoder samples the next token $\mathbf{q}'_t$ from $\mathcal{V}^t$ based on the probability below.

$$
P(\mathbf{q}'_t | \mathbf{q}'_{<t}, \mathbf{q}, \mathcal{V}^t) = \frac{\exp(\mathbf{W}[\mathbf{c_t}; \mathbf{s_t}; \mathbf{q}'_{t-1}] + \mathbf{b})_i}{\sum_{\mathbf{q}'_t \in \mathcal{V}^t} \exp(\mathbf{W}[\mathbf{c_t}; \mathbf{s_t}; \mathbf{q}'_{t-1}] + \mathbf{b})} \tag{4}
$$

where $[\mathbf{c_t}; \mathbf{s_t}; \mathbf{q}'_{t-1}]$ denotes the concatenation of the context $\mathbf{c_t}$, the decoder's hidden state $\mathbf{s_t}$ and the previous output $\mathbf{q}'_{t-1}$.

B. Reinforced Perturbation Policy Learning

Since it is tricky to acquire sufficient effective labeled perturbed workloads to conduct supervised learning [31], we adopt Reinforcement Learning (RL) to train TRAP. The benefit of RL is that it achieves a balance between exploration and exploitation among the numerous perturbation combinations, allowing it to identify the effective ones. Specifically, the encoder-decoder network in Section IV-A acts as an agent that takes action (i.e., generates a SQL token) based on its policy, and the agent receives a reward r to update its policy. The reward is calculated considering the robustness in Definition 3.3, i.e., $r = IUDR$. If the generated workload effectively hinders the index advisor, i.e., leading to the performance drop, then $r > 0$. Otherwise, if the generated workload fails to detect the index advisor's performance loophole, then $r < 0$.

Note that to obtain r, we need a cost metric $c(\mathcal{W}, \mathbf{d}, \mathcal{I})$ to compute $u(\mathcal{W}', \mathbf{d}, f)$. It is infeasible to use the actual runtime

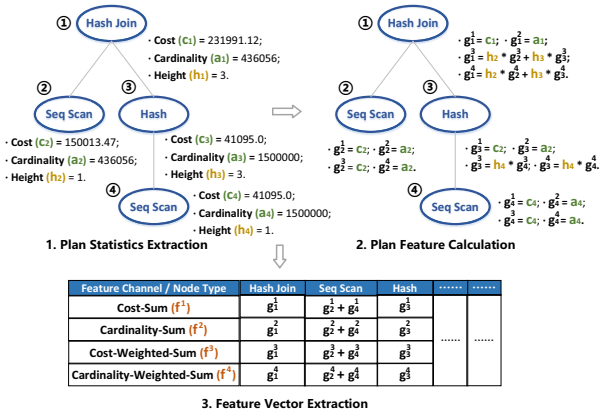

Fig. 4: Example of Feature Modelling on Query Plan.

due to the large overhead of index building and query execution. We can turn to the estimated cost provided by what-if calls. However, we find that the reward is inaccurate due to the estimation error [24]. Inspired by the successful applications of the learned cost model, we **utilize LightGBM [53] to estimate the index utility** $c(\mathcal{W}, \mathbf{d}, \mathcal{I})$, which is both effective with high estimation accuracy and efficient with fast inference [42, 54]. To train LightGBM, we collect a training dataset by randomly generating and executing queries like [19, 36] (more details in Section V), i.e., $\mathcal{D} =< \mathbf{f}, y >$, where $\mathbf{f}$ is a feature vector extracted from the query plan and y is the actual runtime cost.

The feature vector $\mathbf{f}$ is derived as follows. As illustrated in Figure 4, each node in a query plan has several properties: the node type (e.g., "Seq Scan" and "Hash Join"), estimated statistics (e.g., "Cost" and "Cardinality"), and its "Height" in the query plan tree. We define the feature vector as the concatenation of four field vectors, i.e., $\mathbf{f} \in \mathbb{R}^{4\times L}$. Each field vector has the same length, i.e., $\mathbf{f}^l \in \mathbb{R}^L (l = 1, \dots, 4)$, where L is the total number of the possible node types (i.e., operators) in the query plan. Node types that are absent in the query plan will be assigned zero weight in their feature vectors. The four field vectors are: *Cost-Sum* ($\mathbf{f^1}$), *Cardinality-Sum* ($\mathbf{f^2}$), *Cost-Weighted-Sum* ($\mathbf{f^3}$) and *Cardinality-Weighted-Sum* ($\mathbf{f^4}$) respectively. To obtain $\mathbf{f}$, we first extract statistics "Cost" (c_j), "Cardinality" (a_j), and "Height" (h_j) for each node j. Then, for each node j, we calculate the cost (g_j^1), cardinality (g_j^2), weighted-cost (g_j^3), and weighted-cardinality (g_j^4) as follows.

$$g_j^1 = c_j, \quad g_j^2 = a_j,$$
$$g_j^3 = \sum_{\text{k is j's child}} h_k \times g_k^3, \quad g_j^4 = \sum_{\text{k is j's child}} h_k \times g_k^4 \tag{5}$$

Finally, we aggregate the weights with the same node type and obtain the feature vector, i.e., $\mathbf{f}_i^l = \sum_{\text{j's node type is i}} g_j^l$.

We use the learned index utility $y(\mathbf{f})$ to replace $c(\mathcal{W}, \mathbf{d}, \mathcal{I})$ in computing the reward. We adopt the self-critic (SC) method [55] to alleviate the high variance problem by subtracting r with r_b, which is based on the output obtained by greedy search, i.e., choose with the highest probability. We sample a batch of trajectories $\mathcal{B}$ and then average over these

TABLE II: A simplified version of the BNF grammar rules.

SQL ::= SELECT FROM WHERE [GROUPBY] [HAVING] [ORDERBY]
SELECT ::= "**select**" (term (",") term)? | SQL)
FROM ::= "**from**" (table ((",") table)? | ("**join**" table)?) | SQL)
WHERE ::= "**where**" predicate (conjunction predicate)?
predicate ::= column operator (value | SQL)
term ::= column | aggregator "(" column ")"
table ::= <**table**> column ::= <**column**> operator ::= <**operator**>
aggregator ::= <**aggregator**> value ::= <**value**> conjunction ::= <**conjunction**>

trajectories to calculate the policy loss formulated as below.

$$L^{RL} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{q}' \in \mathcal{B}} \sum_t logP(\mathbf{q}'_t | \mathbf{q}'_{<t}, \mathbf{q}, \mathcal{V}^t) \times (r - r_b) \tag{6}$$

where r is calculated based on the learned index utility $y\big(\mathbf{f}(\mathbf{q})\big)$ provided by LightGBM using query $\mathbf{q}$'s feature vector extracted by the above procedure and r_b is calculated based on $\mathbf{q}^g$, which is the query obtained by the greedy search.

C. Index Advisor Independent Pretraining

RL training in Section IV-B explores and exploits the space of possible perturbation combinations but is typically time-consuming due to the large trajectories for a good policy [56]. To enhance the training efficiency, we propose to pre-train `TRAP` prior to RL. This stage is irrelevant to any index advisor, and its goal is to initialize `TRAP` with a better understanding of the SQL (e.g., the overall syntactic structure) before RL.

Specifically, we pre-train `TRAP` based on a synthetic dataset $\mathcal{Q} = \{\mathbf{q}, \mathbf{q}'\}$. For each original query $\mathbf{q}$, we randomly sample and replace tokens to obtain the perturbed query $\mathbf{q}'$ (more details in Section V). Then, the parameters of `TRAP` are updated to optimize the likelihood of generating $\mathbf{q}'$ from $\mathbf{q}$.

$$L^p = -\sum_{(\mathbf{q},\mathbf{q}') \in \mathcal{Q}} \sum_{t<|\mathbf{q}|} logP(\mathbf{q}'_t | \mathbf{q}'_{t-1}, \mathbf{q}, \mathcal{V}^t) \tag{7}$$

Note that although both the encoder and decoder of `TRAP` are pre-trained, we only transfer the parameters of the encoder to the RL stage [57]. The intuition is to enhance `TRAP`'s ability to capture the overall context of the input SQL query. The decoder is refreshed at the beginning of RL to explore various perturbations and receive index-relevant signals directly.

D. Constraint-Aware Reference Tree

Although the encoder-decoder structure [51] has been widely applied in many NLP tasks, it fails to generate a valid perturbed query that meets the given perturbation constraints (Note that the constrained decoding technique adopted in the SQL generation tasks, e.g., the NL2SQL [58] only guarantees the SQL grammar). Intuitively, since SQL is well-structured and each token can be chosen from a limited set of tokens in the vocabulary, our solution is to construct a `Constraint-Aware Reference Tree` for each SQL query and locate *the legitimate vocabulary* at each step by traversing the tree.

Construction of Constraint-Aware Reference Tree. Given an input query $\mathbf{q}$, we initialize G based on the Backus–Naur Form (BNF) grammars [59]. Each BNF rule defines a non-terminal symbol recursively by a set of terminals (i.e., query tokens) or other non-terminals. A simplified version of BNF

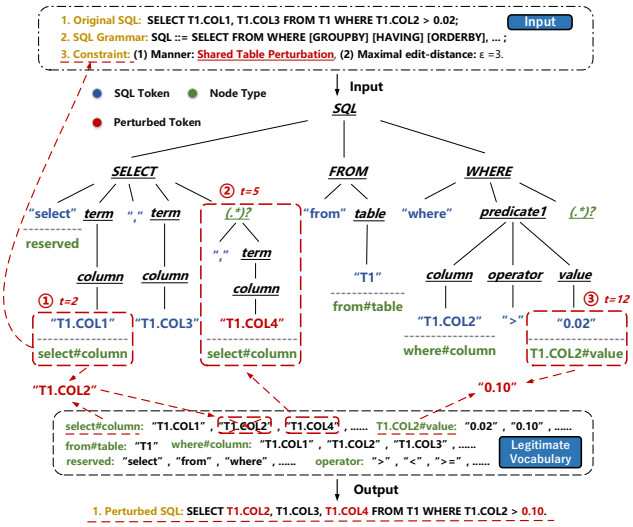

Fig. 5: Illustration of Constraint-Aware Reference Tree.

rules are illustrated in Table II. Consequently, as shown in Figure 5, G's root node r corresponds to the SQL statement, a leaf node l corresponds to an actual SQL token in $\mathbf{q}$, and a non-leaf node n corresponds to a non-terminal symbol in the BNF rules. Such a tree structure can support complex queries when a sub-query is represented as a sub-tree of G for nested queries.

For each node $i \in G$, a node type p^i is assigned. There are three categories of node types: (1) For a non-leaf node n, its node type p^n is equivalent to the non-terminal symbol in the BNF rule; (2) Each leaf node l is assigned a node type based on the clause and its token type[3]; (3) To append additional sub-classes to the query, a child node is constructed at the end of each clause in G and a special node type, i.e., `(.*)?` is assigned. For example, in Figure 5, `select` and `T1.COL1` are the two leaf nodes under the node `SELECT` and their node types are `reserved` and `select#column` respectively. The node with a special type, i.e., `(.*)?` serves as the last child node in each clause, i.e., the node `SELECT` and `WHERE`.

To reduce the storage cost, we maintain a global vocabulary $\mathcal{V}$ for G, which is segmented into several regions. We use the leaf node l's node type p^l to locate a region $\mathcal{V}^{p^l}$ of the legitimate tokens for l. Each region in the vocabulary is instantiated based on the node type's legitimate tokens. As shown in Figure 5, the vocabulary contains a set of reserved SQL keywords for node type `reserverd`. For `from#table`, tables in the current dataset $\mathbf{d}$ are included. Legitimate tokens for predicate values are sampled from the current dataset and workloads. For example, for `T1.COL2#value`, we sample the values of column `COL2` in table `T1` or extract values from the predicates in the workloads in Section V-A.

Masking Invalid Tokens Based on Constraint-Aware Reference Tree. After G is initialized and $\mathcal{V}$ is instantiated, while generating $\mathbf{q}'$, we dynamically mask invalid tokens in $\mathcal{V}$ and update G. Algorithm 1 depicts the overall process. Starting from the first token **(line 1)**, the perturbed query $\mathbf{q}'$ is

[3]Due to space constraints, more information about the BNF rules, token types, and vocabulary is presented in:

Algorithm 1: Generation of $\mathbf{q}'$ based on G

Input: Original queries $\mathbf{q}$, Constraint-Aware Reference Tree G, Perturbation constraint, Edit distance ϵ
Output: Perturbed queries $\mathbf{q}'$

1 $t = 1$
2 **while do**
3 **if** *current edit distance* $k(\mathbf{q}, \mathbf{q}') < \epsilon$ *and (token type can be modified or node* t *is* `(.*)?` *)* **then**
4 Obtain the legitimate vocabulary $\mathcal{V}^{p^t}$
5 Sample $\mathbf{q}'_t$ from $\mathcal{V}^{p^t}$ by $P(\mathbf{q}'_t|\mathbf{q}'_{<t}, \mathbf{q}, \mathcal{V}^t)$
6 **if** $\mathbf{q}_t \neq \mathbf{q}'_t$ **then**
7 Update current edit distance $k(\mathbf{q}, \mathbf{q}')$
8 **for** *each leaf node* i *behind* t *in* G *affected by* $\mathbf{q}'_t$ **do**
9 Update p^i and $\mathcal{V}^{p^i}$
10 t++

generated in a token-by-token manner **(line 2)**. At step t, we first check if the edit distance has not been exceeded and the modification is allowed in the perturbation constraints, then a new token can be appended **(line 3)**. Figure 5 presents an example under the `Shared Table Perturbation` with $\epsilon = 3$, "`T1.COL1`" can be altered and a new column, i.e., `T1.COL4` can be added to `(.*?)`. Then we traverse to the leaf node t and locate the legitimate vocabulary $\mathcal{V}^{p^t}$ of its node type **(line 4)**. We sample a token $\mathbf{q}'_t$ from the legitimate vocabulary $\mathcal{V}^{p^t}$ based on the probability calculated in Equation 4 **(line 5)**. If the sampled token $\mathbf{q}'_t$ is different from the original token, then we update the edit distance **(line 6)**. We also look ahead to all the nodes that will be affected by the current token and update their node types and legitimate vocabulary **(line 8)**. For instance, "`T1.COL1`" in Figure 5 is replaced by "`T1.COL2`", then "`T1.COL2`" is masked in $\mathcal{V}^{\texttt{select\#column}}$ to avoid the repetitive occurrence of columns in the same clause. The node type `T1.COL2#value` for node "`0.02`" will be updated by `?#value`, where `?` is a placeholder for a column in the predicate and is instantiated by "`T1.COL2`" since "`T1.COL2`" remains unchanged.

V. Experiments

In this section, we conduct experiments to answer two research questions. (1) Can current index advisors deliver high and stable performance over the perturbation-based workloads (Section V-B)? (2) Are the components of `TRAP` effective in generating perturbation-based adversarial workloads that successfully incur performance drops of the index advisors (Section V-C)? Furthermore, we present an in-depth **analysis and discoveries** about these index advisors in Section VI.

A. Experimental Setup

Index Advisors. As shown in Table III, we assess the robustness of ten index advisors, including heuristic-based index advisors, i.e., Extend [3], DB2Advis [4], AutoAdmin [5], Drop [6], Relaxation [29] and DTA [11]; learning-based index

TABLE III: Index advisors include heuristic-based and learning-based methods that select single-column (S) and multi-column (M) indexes within the storage and #index constraints, using different selection criteria, strategies etc.

Heuristic-based Index Advisors				
Victim	**Constraint**	**Type**	**Criterion**	**Strategy**
Extend [3]	Storage	S/M	$\frac{Cost}{Storage}$	Incremental
DB2Advis [4]	Storage	S/M	$\frac{Cost}{Storage}$	Incremental
AutoAdmin [5]	#index	S/M	$Cost$	Incremental
Drop [6]	#index	S	$Cost$	Decremental
Relaxation [29]	Storage	S/M	$\frac{Cost}{Storage}$	Decremental
DTA [11]	Storage	S/M	$Cost$	Incremental
Learning-based Index Advisors				
Victim	**Constraint**	**Type**	**Learning**	$\mathcal{I}^b$
SWIRL [7]	Storage	S/M	PPO	Extend
DRLindex [9]	#index	S	DQN	Drop
DQN [8]	#index	S/M	DQN	AutoAdmin
MCTS [10, 60]	#index	S/M	MCTS	AutoAdmin

advisors, i.e., SWIRL [7], DRLindex [9, 13], DQN [8] and MCTS [10, 60]. According to Table III, these index advisors cover a wide range of variety, including different tuning constraints (i.e., storage budget or #index), index types (i.e., single and multi-column indexes), selection criteria and strategies, and the underlying reinforcement learning methods.

Datasets. (1) TPC-H is an open-source OLAP benchmark that contains 8 tables and 61 columns; (2) TPC-DS is an open-source OLAP benchmark that contains 25 tables and 429 columns; (3) TRANSACTION is a real-world OLTP benchmark of banking that contains 10 tables and 189 columns.

Queries. For TRANSACTION, we adopt the queries from real-world transactions. For TPC-H and TPC-DS, we follow the method in [19, 36] to enrich the diversity of queries, which synthesizes additional Select-Project-Aggregate-Join (SPAJ) queries according to a meaningful join graph.

Workloads. We randomly sample from the queries and construct workloads with random sizes in [1, 50]. Since not all the index advisors have explicitly considered the query frequency, we assign a unit frequency, i.e., 1, to each query in the workload for a fair assessment. However, the frequency is implicitly considered if multiple identical queries appear in the workload. Besides, our framework can support different frequencies with little effort by multiplying the reward with the frequency in Equation 6. We construct 20,000 workloads and randomly perturb them to pre-train `TRAP` (Note that this process is an offline one-time effort without the interaction with specific index advisor. The comparison among other pre-trained models are presented in Section V-C). We construct another 5,000 workloads to train the learning-based index advisors and the learned cost model, i.e., LightGBM in Section IV-B. Then, we split the 5,000 workloads into the training/testing/validation sets (8:1:1) to train and evaluate `TRAP`. According to Definition 3.3, if the utility of an index advisor $u(\mathcal{W}, \mathbf{d}, f) > \theta$, i.e., high performance for a workload $\mathcal{W}$ in the set, then we generate $\mathcal{W}'$ by `TRAP` or other competitors and compute *IUDR* based on $u(\mathcal{W}, \mathbf{d}, f)$ and $u(\mathcal{W}', \mathbf{d}, f)$. Note that we detect and exclude all the potential non-sargable workloads from $\mathcal{W}'$, i.e., $u(\mathcal{W}', \mathbf{d}, f) < \theta$ for all the index advisors (Definition 3.3), which are not in the region of the assessment, and the analysis over these workloads is in Section VI-C.

Evaluation Metric. Given the strictly same tuning constraints (e.g., the same storage budget) provided in the original paper, the evaluation metric is *IUDR*. Similar to the focus of previous studies, it measures the relative cost reduction, and a higher *IUDR* means a larger performance drop (i.e., less robust)[4]. In computing the index utility u in Definition 3.2, the relative cost reduction of the index advisor is compared with a baseline configuration $\mathcal{I}^b$. For heuristic-based index advisors, $\mathcal{I}^b$ is the null set, i.e., using no index at all. For learning-based index advisors, $\mathcal{I}^b$ is given by a heuristic-based index advisor since learning-based index advisors are claimed to outperform heuristic-based index advisors [7, 8]. In particular, because the performance is affected by the tuning constraints and the index type (i.e., the single-column and multi-column index), a fair baseline configuration is chosen within the same constraint and index type. For example, as shown in Table III, the baseline for SWIRL is Extend because they meet the same storage tuning constraint and support multi-column indexes. We repeat the overall process three times and report the average *IUDR*.

Implementation. All the experiments are conducted with Python 3.7, PostgreSQL 12.5 on a workstation with two Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz, 256 GB main memory, and a GeForce RTX 2080 Ti graphics card. We leverage the open-source implementation of heuristic-based index advisors [2], SWIRL [7] and DQN [8]. We implement DRLindex [9] and the UCT version of MCTS [60] and train all the learning-based index advisors according to the details illustrated in the original paper. Unless stated, the default initial index utility is $\theta = 0.1$, and the maximum edit distance allowed is $\epsilon = 5$ (the choice of the threshold is discussed in Section V-C). For `TRAP`, the embedding size of GRU is set to be 128, the learning rate is 0.001, and it is trained for 200 epochs in pre-training and 100 epochs in reinforcement learning. We use feature normalization, implement log-transformation [61] and minimize the Mean Square Error to train LightGBM.

B. Robustness Assessment of Index Advisors

We generate perturbation-based adversarial workloads $\mathcal{W}'$ with four different methods. (1) `Random`: Randomly replace tokens of the queries in the original workload to synthesize perturbed queries. (2) `GRU`: Generate the perturbed queries in a token-by-token manner using a GRU layer. (3) `Seq2Seq`: Generate the perturbed queries by a vanilla Seq2Seq model, where the encoder is the Bi-GRU layer and the decoder is the GRU layer. (4) `TRAP`: The method described in Section IV. All the above methods are equipped with the same tree structure to guarantee the validity of queries, and `Random` is allowed to generate $5\times$ more perturbed queries to verify its capability.

From Figure 6, we observe that **both heuristic-based and learning-based index advisors are vulnerable to the perturbation-based adversarial workloads generated by**

[4]We get the final indexes returned by each index advisor and observe that the time budget does not exhibit variations spanning several orders of magnitude under the same tuning constraint.

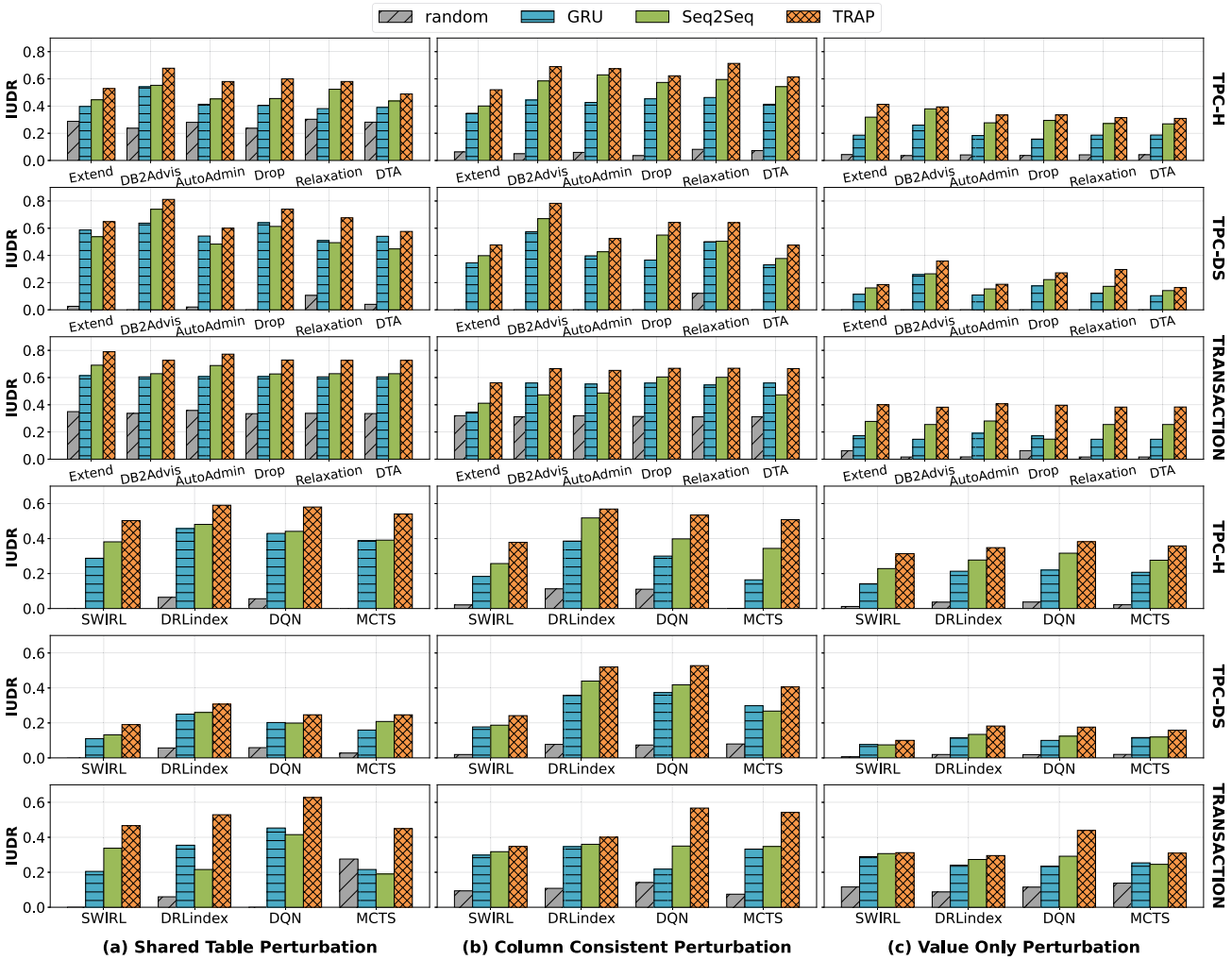

Fig. 6: *IUDR* of index advisors on workloads generated by different methods (A higher *IUDR* means a larger performance drop, i.e., less robust).

TRAP. Specifically, *IUDR* is on average 0.4946 on TPC-H, 0.3893 on TPC-DS, and 0.5177 on TRANSACTION under the three perturbation constraints. SWIRL is generally the most robust learning-based index advisor. This might be attributed to the most fine-grained state representation adopted in the workload modelling and the invalid action masking strategy over the action space of candidate indexes (More detailed analysis is provided in Section VI-A).

Comparison among different workload generation methods verifies **the necessity of designing a workload generation framework for robustness assessment**. A naive workload generation method, e.g., the `random` method, cannot accurately measure the index advisors' robustness. Its *IUDR* is barely noticeable (*IUDR* < 0.005) over many index advisors on the TPC-DS benchmark. For example, it can not distinguish the performance of Extend, DB2Advis, AutoAdmin, and Drop on the TPC-DS benchmark. Furthermore, the values of *IUDR* are inconsistent with other workload generation methods. For example, other methods achieve a larger *IUDR* under `Column Consistent Perturbation` than `Value Only Perturbation`, while `random` can not differentiate the two perturbations, i.e., the resulting *IUDR* does not show significant differences among these perturbations. TRAP *outperforms other workload generation competitors*. The reasons are two-fold: (1) the attention mechanism adopted in TRAP better captures the overall SQL context and facilitates the exploration over more effective perturbation combinations; (2) the proposed pre-training step relieves TRAP from learning a basic understanding about the SQL in RL and therefore TRAP can focus on the index-sensitive parts (e.g., the critical predicates in the WHERE clause) and improves the training effectiveness (more details in Section V-C). *Index advisors are less robust to perturbed queries with more flexible perturbations*. We can observe that all the workload generation methods under `Shared Table Perturbation` and `Column Consistent Perturbation` achieve a higher *IUDR* than under `Value Only Perturbation`. For `Shared Table Perturbation`, TRAP, `Seq2Seq` and `GRU` achieves 88.56%, 95.59% and 143.95% higher *IUDR* than `Value Only Perturbation` on average. For `Column Consistent Perturbation`, a 89.62%, 98.41% and 121.44% higher *IUDR* are obtained by TRAP, `Seq2Seq` and `GRU` on average. The underlying reason is that the first two perturbation constraints are more flexible (e.g., perturbations over critical columns other than values). Accordingly, the workload generation methods can explore a

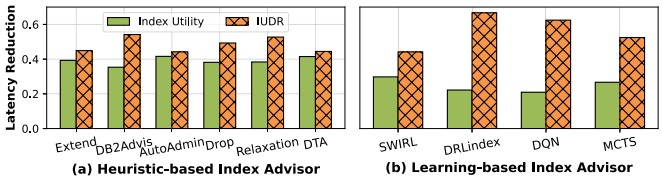

Fig. 7: The relation between index utility and robustness metric *IUDR*.

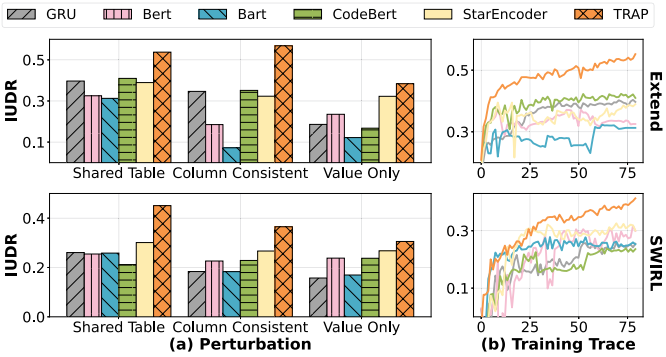

Fig. 8: Performance with different generation modules.

larger space of perturbations on more index-sensitive parts of a SQL (e.g., the WHERE clause) and degrade the performance of the index advisors more severely.

We further investigate the relation between *IUDR* and the absolute index utility after perturbation. As shown in Figure 7, we observe that for both the heuristic-based and learning-based index advisors, a higher *IUDR* typically corresponds to a lower index utility (i.e., inferior performance) after perturbation. Thus, if we already know the index utility of each index advisor on the original workload, the value of *IUDR* can reflect the end-to-end performance of index advisors to a large degree.

C. Ablation Study and Impacts of Parameters

Ablation on Generation Module. To investigate the impact of the generation module of TRAP, we consider the pre-trained language model (PLM) [62, 63] and implement the following variants: (1) `GRU`: The encoder is a GRU layer and the decoder is removed; (2) `Bert`: The encoder is equipped with the transformer-based pre-trained model Bert [64]; (3) `Bart`: The pre-trained bart [65] (transformer encoder-decoder model) is adopted for the generation; (4) `CodeBert`: The bert-based model pre-trained with the large bi-modal data (documents & code) corpus [66]; (5) `StarEncoder`: The encoder is equipped with a recent transformer-based model [67] trained on 80+ programming languages. We report the *IUDR* and the training trace of Extend and SWIRL on the TPC-H benchmark.

From Figure 8, we can see that *the proposed module of* TRAP *does improve the effectiveness of the workload generation.* Specifically, TRAP achieves a 41.04% higher *IUDR* with $1392.19\times$ fewer parameters compared with other competitors on average. Nevertheless, the pre-trained models, which adopt a more advanced architecture, fail to achieve a higher *IUDR*. The inferior performance of the pre-trained models might be attributed to the following reasons: (1) The large-scale transformer architecture needs to be further optimized when applied to the RL paradigm [68, 69], which is sensitive to the underlying model design [70] and typically requires a large

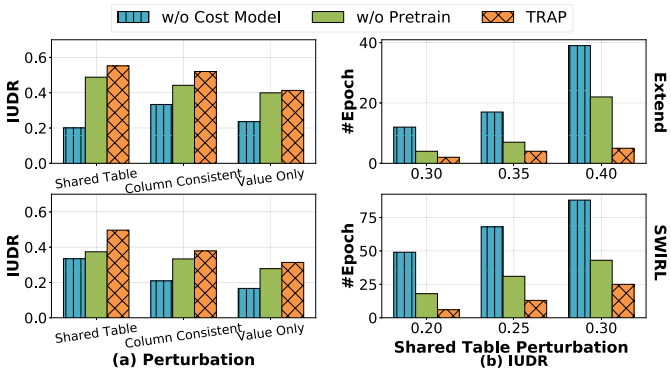

Fig. 9: Performance under different training paradigms.

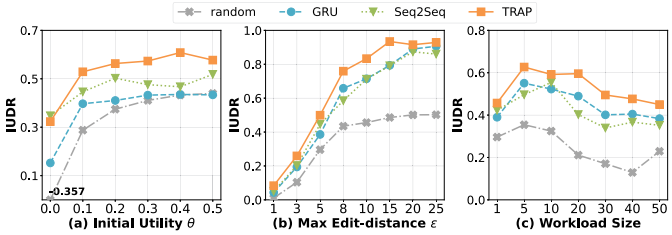

Fig. 10: *IUDR* v.s. initial utility θ, edit distance ϵ and workload size $|\mathcal{W}|$.

number of samples to learn a good policy; (2) These models are typically pre-trained on a generic corpus deviating greatly from the SQL, which needs a large effort to deal with the domain adaption issue [71] during the fine-tune process.

Ablation on Training Paradigm. We investigate the impact of the presented training paradigms. Specifically, w e implement two training paradigms: (1) `w/o Cost Model`: In the RL training, use the what-if calls to acquire the estimated cost; (2) `w/o Pretrain`: Train TRAP using only reinforcement learning. As displayed in Figure 9 (a), without the learned index utility model[5], the *IUDR* is 47.46% and 41.46% worse on average. It means that the cost estimation model provides a more accurate estimation than the what-if optimizer [24], to guide the perturbation procedure of TRAP. In Figure 9 (b), without pre-training, it takes more than $2.72\times$ and $2.37\times$ more epochs to reach a desired value of *IUDR* for SWIRL and Extend on average. The underlying reason is that pre-training transfers the understanding of SQL from supervised learning and thus boosts the training efficiency o f RL.

Impact of Hyper-parameters. We investigate the impacts of parameters on the robustness assessment of index advisors, i.e., (1) the initial index utility threshold θ, (2) the maximal edit distance allowed ϵ, and (3) the workload size $|\mathcal{W}|$. The experiment is conducted under `Shared Table Perturbation` against Extend on the TPC-H benchmark. Figure 10 (a) reports *IUDR* with respect to θ from 0.0 to 0.5. *For an index advisor, its robustness is positively correlated with its performance on the original workload.* The *IUDR* of all the methods increases as θ increases. The underlying reason is the difficulty of deteriorating a poorly performed algorithm. `random` method fails to generate workloads that deteriorate the performance of Extend (i.e., $IUDR < 0$). Next, we conduct

[5]We presents the end-to-end performance, i.e., the *IUDR* due to space limits, the estimation accuracy is provided in:

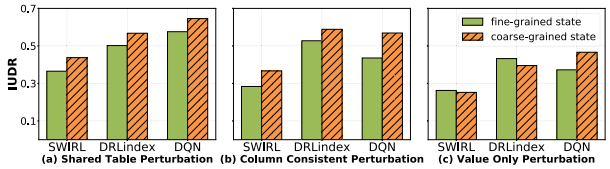

Fig. 11: *IUDR* vs. the adopted state representations.

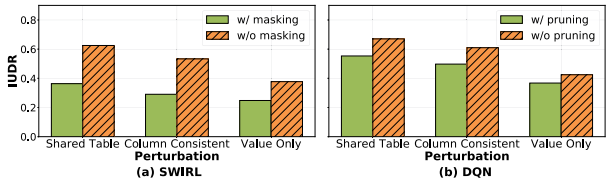

Fig. 12: *IUDR* vs. the candidate pruning in the action space.

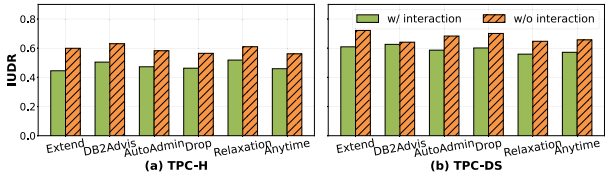

Fig. 13: *IUDR* vs. the consideration of the index interaction.

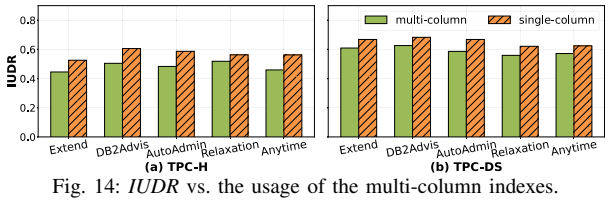

Fig. 14: *IUDR* vs. the usage of the multi-column indexes.

perturbation with different amplitudes. Figure 10 (b) reports *IUDR* with different values of edit distance ϵ. We observe that *the robustness is dependent on the amplitude of perturbation*. A smaller ϵ produces workloads that are more similar to the original workload, and thus the index advisor is more likely to perform well. We finally generate workloads of a fixed size ranging from 1 to 50 and report *IUDR* in Figure 10 (c). It shows that TRAP outperforms other methods on the multi-query workloads with different workload sizes.

VI. Analysis and Discovery

In this section, we analyze the robustness of index advisors from the following aspects: (1) the impact of module designs on the robustness of learning-based index advisors (Section VI-A); (2) the impact of strategy choices on the robustness of heuristic-based index advisors (Section VI-B); (3) the effect of query changes (Section VI-C).

A. Learning-based Index Advisors

Our first discovery is that **the granularity of state representation affects the robustness of learning-based index advisors**. From the study in Section II, we choose two typical state representations: (1) `fine-grained state` in SWIRL [7], which captures the workload characteristics, including the operators and estimated cost extracted from the query plans and the corresponding frequency; (2) `coarse-grained state` in DRLindex [9], which adopts a matrix to indicate the existence of columns in the workload and an access vector to count the total occurrence of these columns. Then we change the state representations on three index advisor backbones. We only replace the state representations with these two types while keeping the rest of the backbones fixed. We use TRAP to generate adversarial workloads and report *IUDR* in Figure 11. We can see that index advisors with the `coarse-grained state` are more vulnerable to adversarial workloads. For example, all index advisors show significantly higher *IUDR* (14.41% and 22.20% higher on average) under `Shared Table Perturbation` and `Column Consistent Perturbation` respectively.

The exception is under `Value Only Perturbation`, where not all the index advisors achieve smaller *IUDR* with the `fine-grained state`. This is attributed to the failure of these two states to capture `Value Only Perturbation`. The `coarse-grained state` only considers the existence of columns in the workload, and the `fine-grained state` [7] only considers the operators in the query plans. For `Value Only Perturbation`, the optimizer will choose the same query plan, i.e., the same set of operators regardless of the value changes, and thus the `fine-grained state` representation can not capture such changes.

Our second discovery is that **candidate pruning in action space affects the robustness of learning-based index advisors**. Both SWIRL and DQN adopt the candidate pruning techniques in the action space. SWIRL [7] adopts invalid action masking [72] to remove invalid candidates that (1) are syntactically irrelevant to the workload, (2) exceed the storage budget, (3) are selected already, and (4) meet the invalid precondition principle. DQN [8] classifies the columns considering the syntactic structure and leverages five heuristic rules to generate promising candidates.

We report the *IUDR* of index advisors with different pruning techniques in Figure 12. We observe that both SWIRL and DQN are more vulnerable to adversarial workloads without candidate pruning in the action space. For example, SWIRL's *IUDR* without candidate pruning is 69.18% higher, and DQN's is 19.72% higher on average.

B. Heuristic-based Index Advisors

We divide the index advisors into two groups based on tuning constraints (i.e., storage or #index) and compare the number of sub-optimal solutions over all the adversarial workloads generated in Section V. We find that Extend [3] and Drop [6] are the worst performers in each group.

We find that Extend is inferior since it neglects the index interaction sometimes due to the predefined heuristic. Index interaction [73] refers to the phenomenon that the benefit of one index can be affected by the presence of another index. For instance, suppose creating index A or B independently does not impact the given workload, but a significantly positive impact can be obtained with these two indexes built together.

We conduct the following experiment to verify that **neglecting the index interaction hurdles the robustness of heuristic-based index advisors**. Specifically, we modify the implementation of heuristic-based index advisors and utilize two methods to calculate the benefit of multiple indexes during the selection process. (1) `w/ interaction`: calculate the

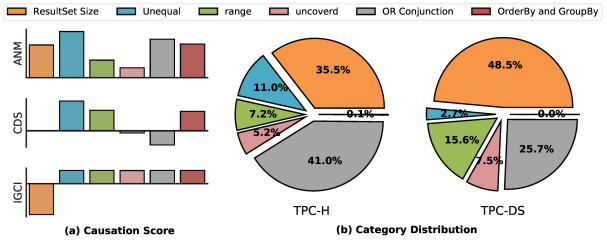

Fig. 15: Effect of six types of query changes.

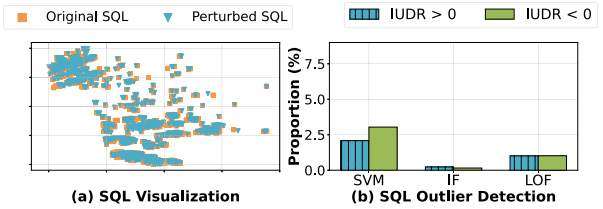

Fig. 16: Visualization of SQL vectors before/after perturbation and the proportion of outliers detected in perturbations with $IUDR > 0$ and $IUDR < 0$.

benefit with all the indexes built; (2) `w/o interaction`: calculate the benefit of every single index with them built independently and then obtain the benefit of multiple indexes by averaging the benefits over all the indexes. As shown in Figure 13, a smaller *IUDR* is achieved if they consider the index interaction when calculating the index benefits.

Apart from the index interaction, we notice that **the index type, i.e., the single-column and multi-column indexes, also have an impact on the robustness**. Compared with AutoAdmin, Drop only recommends single-column indexes. To verify the impact of the multi-column indexes, we modify the implementation of different heuristic-based index advisors and make them consider the index candidates with (1) only the `single-column` indexes and (2) also the `multi-column` indexes. As shown in Figure 14, all the index advisors exhibit a smaller *IUDR* if multi-column indexes are considered.

C. Impact of Query Change

We study types of SQL changes that are considered to be factors relevant to the performance of index[6]: (1) the `Resultset Size` has been dramatically enlarged after perturbation; (2) an operator is changed to `Unequal` operator "$\neq$"; (3) change the operator "$=$" to a `range` (i.e., "$\geq, \leq, <, >$"); (4) columns in the SELECT clause are `uncovered` in the WHERE clause after perturbation; (5) the conjunction is replaced by `OR Conjunction`; (6) change the columns in `ORDER BY and GROUP BY` to enlarge the discrepancy.

We find **these types of SQL changes can make a query non-sargable.** A sargable query [43, 44] means that the DBMS engine can take advantage of an index to speed up its execution. To verify our discovery, we first use causal models [74] to determine whether there is a causal relationship between the aforementioned types and *IUDR*. A causal model computes a causation score between two random variables, i.e., X and Y. If the causation score is positive, it means that X is a cause that leads to the effect of Y. We collect pairs of (x, y) from our experiments to compute the causation score. If a perturbed workload $\mathcal{W}'$ has the following property: all the index advisors have a small index utility on it, i.e., $\forall f, u(\mathcal{W}', \mathbf{d}, f) < \theta$, it means $\mathcal{W}'$ is non-sargable. Then we construct the pair (x, y), where x is the occurrence of one of six query change types in the perturbed workload $\mathcal{W}'$ and y is the *IUDR*. We utilize three causal models in [75]. As shown in Figure 15 (a), *most causal models agree that the above SQL change types cause the index utility to decrease*, i.e., the causation scores are positive. Moreover, we count the number of queries associated with these query changes. Figure 15 (b) shows the distribution of six categories of changes in the non-sargable queries. We can see that a large proportion ($> 70\%$) of non-sargable queries are results of changing to `OR Conjunction` or expanding the `ResultSet Size`.

Finally, we find that workload $\mathcal{W}'$ generated by `TRAP` affect the robustness of index advisors, **not because $\mathcal{W}'$ are Out-of-Distribution (OOD) samples.** We first visualize the representation vector of queries obtained by `TRAP`'s encoder before and after perturbation with t-SNE [76]. As shown in Figure 16 (a), the original and output queries are indistinguishable and follow the same distribution. Moreover, we mix the original and perturbed queries and use anomaly detection algorithms [77, 78, 79] to detect outlier queries. As shown in Figure 16 (b), the percentage of outlier queries in effective perturbations ($IUDR > 0$) and ineffective perturbations ($IUDR < 0$) are similar. The dominant fraction (i.e., $97\% \sim 99\%$) of effective perturbed queries are "normal" queries. Thus, the adversarial workloads are effective not because they are dissimilar to the original queries, but because they capture the potential loopholes of the index advisors.

VII. Conclusion

We propose a framework that automatically generates perturbation-based workloads to assess the robustness of existing index advisors. We conduct comprehensive robustness assessments of ten index advisors on various benchmarks. We provide insightful discoveries for both heuristic-based and learning-based index advisors, which could facilitate the design of future robust index advisors.

Summarized Findings. Our findings suggest that: (1) Perturbation-based adversarial workloads are effective to assess the robustness of the index advisor because they do not deviate too much from the original workloads but can identify the performance loopholes due to the workload drifts in practice; (2) To design a more robust learning-based index advisor, it is beneficial to adopt a fine-grained state representation to capture the workload characteristics and a candidate pruning strategy in the action space to prune syntactic irrelevant or useless candidates in advance; (3) To design a more robust heuristic-based index advisor, it is vital to take the index interaction into account during the selection process and consider the usage of the multi-column indexes. To increase real-world applicability on more indexes advisors and workloads, we have implemented `TRAP` in the open-source database openGauss[7].

[6]https://docs.oracle.com/cd/B19306_01/server.102/b14211/data_acc.htm

REFERENCES

[1] R. M. Perera, B. Oetomo, B. I. P. Rubinstein, and R. Borovica-Gajic, "DBA bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees," in *ICDE*, 2021, pp. 600–611.

[2] J. Kossmann, S. Halfpap, M. Jankrift, and R. Schlosser, "Magic mirror in my hand, which is the best in the land? an experimental evaluation of index selection algorithms," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2382–2395, 2020.

[3] R. Schlosser, J. Kossmann, and M. Boissier, "Efficient scalable multi-attribute index selection using recursive strategies," in *ICDE*, 2019, pp. 1238–1249.

[4] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman, and A. Skelley, "DB2 advisor: An optimizer smart enough to recommend its own indexes," in *ICDE*, 2000, pp. 101–110.

[5] S. Chaudhuri and V. R. Narasayya, "An efficient cost-driven index selection tool for microsoft SQL server," in *VLDB*, 1997, pp. 146–155.

[6] K. Whang, "Index selection in relational databases," *Foundations of Data Organization*, pp. 487–500, 1987.

[7] J. Kossmann, A. Kastius, and R. Schlosser, "SWIRL: selection of workload-aware indexes using reinforcement learning," in *EDBT*, 2022, pp. 2:155–2:168.

[8] H. Lan, Z. Bao, and Y. Peng, "An index advisor using deep reinforcement learning," in *CIKM*, 2020, pp. 2105–2108.

[9] Z. Sadri, L. Gruenwald, and E. Leal, "Drlindex: deep reinforcement learning index advisor for a cluster database," in *IDEAS*, 2020, pp. 11:1–11:8.

[10] X. Zhou, L. Liu, W. Li, L. Jin, S. Li, T. Wang, and J. Feng, "Autoindex: An incremental index management system for dynamic workloads," in *ICDE*, 2022, pp. 2196–2208.

[11] S. Chaudhuri and V. Narasayya., "Anytime algorithm of database tuning advisor for microsoft sql server." https://www.microsoft.com/en-us/research/publication/, 2020.

[12] A. Kane., "Dexter - the automatic indexer for postgres." https://github.com/ankane/dexter, 2017.

[13] Z. Sadri, L. Gruenwald, and E. Leal, "Online index selection using deep reinforcement learning for a cluster database," in *ICDE Workshops*, 2020, pp. 158–161.

[14] R. M. Perera, B. Oetomo, B. I. P. Rubinstein, and R. Borovica-Gajic, "HMAB: self-driving hierarchy of bandits for integrated physical database design tuning," *Proc. VLDB Endow.*, vol. 16, no. 2, pp. 216–229, 2022.

[15] S. Das, M. Grbic, I. Ilic, I. Jovandic, A. Jovanovic, V. R. Narasayya, M. Radulovic, M. Stikic, G. Xu, and S. Chaudhuri, "Automatically indexing millions of databases in microsoft azure SQL database," in *SIGMOD Conference*, 2019, pp. 666–679.

[16] Y. Zhu, Y. Tian, J. Cahoon, S. Krishnan, A. Agarwal, R. Alotaibi, J. Camacho-Rodríguez, B. Chundatt, A. Chung, N. Dutta, A. Fogarty, A. Gruenheid, B. Haynes, M. Interlandi, M. Iyer, N. Jurgens, S. Khushalani, B. Kroth, M. Kumar, J. Leeka, S. Matusevych, M. Mittal, A. Müller, K. Muthyala, H. Nagulapalli, Y. Park, H. Patel, A. Pavlenko, O. Poppe, S. Ravindran, K. Saur, R. Sen, S. Suh, A. Tarafdar, K. Waghray, D. Wang, C. Curino, and R. Ramakrishnan, "Towards building autonomous data services on azure," in *SIGMOD Conference Companion*, 2023, pp. 217–224.

[17] P. Negi, Z. Wu, A. Kipf, N. Tatbul, R. Marcus, S. Madden, T. Kraska, and M. Alizadeh, "Robust query driven cardinality estimation under changing workloads," *Proc. VLDB Endow.*, vol. 16, no. 6, pp. 1520–1533, 2023.

[18] B. Li, Y. Lu, and S. Kandula, "Warper: Efficiently adapting learned cardinality estimators to data and workload drifts," in *SIGMOD Conference*. ACM, 2022, pp. 1920–1933.

[19] P. Negi, R. C. Marcus, A. Kipf, H. Mao, N. Tatbul, T. Kraska, and M. Alizadeh, "Flow-loss: Learning cardinality estimates that matter," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2019–2032, 2021.

[20] K. Vaidya, A. Dutt, V. R. Narasayya, and S. Chaudhuri, "Leveraging query logs and machine learning for parametric query optimization," *Proc. VLDB Endow.*, vol. 15, no. 3, pp. 401–413, 2021.

[21] J. Zhou, N. Bruno, M. Wu, P. Larson, R. Chaiken, and D. Shakib, "SCOPE: parallel databases meet mapreduce," *VLDB J.*, vol. 21, no. 5, pp. 611–636, 2012.

[22] J. Wang, T. Li, A. Wang, X. Liu, L. Chen, J. Chen, J. Liu, J. Wu, F. Li, and Y. Gao, "Real-time workload pattern analysis for large-scale cloud databases," *arXiv Preprint*, vol. https://arxiv.org/abs/2307.02626, 2023.

[23] A. Aleyasen, M. Morcos, L. Antova, M. Sugiyama, D. Korablev, J. Patvarczki, R. Mutreja, M. Duller, F. M. Waas, and M. Winslett, "Intelligent automated workload analysis for database replatforming," in *SIGMOD Conference*, 2022, pp. 2273–2285.

[24] V. Leis, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann, "How good are query optimizers, really?" *Proc. VLDB Endow.*, vol. 9, no. 3, pp. 204–215, 2015.

[25] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," in *CIDR*. www.cidrdb.org, 2019.

[26] B. Ding, S. Chaudhuri, J. Gehrke, and V. R. Narasayya, "DSB: A decision support benchmark for workload-driven and traditional database systems," *Proc. VLDB Endow.*, vol. 14, no. 13, pp. 3376–3388, 2021.

[27] Y. Han, Z. Wu, P. Wu, R. Zhu, J. Yang, L. W. Tan, K. Zeng, G. Cong, Y. Qin, A. Pfadler, Z. Qian, J. Zhou, J. Li, and B. Cui, "Cardinality estimation in DBMS: A comprehensive benchmark evaluation," *Proc. VLDB Endow.*, vol. 15, no. 4, pp. 752–765, 2021.

[28] M. Kurmanji and P. Triantafillou, "Detect, distill and update: Learned DB systems facing out of distribution data," *arXiv Preprint*, 2022. [Online].

Available: https://arxiv.org/abs/2210.05508
[29] N. Bruno and S. Chaudhuri, "Automatic physical database tuning: A relaxation-based approach," in *SIGMOD Conference*, 2005, pp. 227–238.
[30] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998.
[31] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nat.*, vol. 521, no. 7553, pp. 436–444, 2015.
[32] A. Sharma, F. M. Schuhknecht, and J. Dittrich, "The case for automatic database administration using deep reinforcement learning," *arXiv Preprint*, 2018. [Online]. Available: https://arxiv.org/abs/1801.05643
[33] D. R. Slutz, "Massive stochastic testing of SQL," in *VLDB*. Morgan Kaufmann, 1998, pp. 618–622.
[34] M. Rigger and Z. Su, "Finding bugs in database systems via query partitioning," *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, pp. 211:1–211:30, 2020.
[35] X. Liu, X. Kong, L. Liu, and K. Chiang, "Treegan: Syntax-aware sequence generation with generative adversarial networks," in *ICDM*. IEEE Computer Society, 2018, pp. 1140–1145.
[36] L. Zhang, C. Chai, X. Zhou, and G. Li, "Learnedsqlgen: Constraint-aware SQL generation using reinforcement learning," in *SIGMOD Conference*, 2022, pp. 945–958.
[37] Q. Zhou, J. Arulraj, S. B. Navathe, W. Harris, and J. Wu, "SIA: optimizing queries using learned predicates," in *SIGMOD Conference*, 2021, pp. 2169–2181.
[38] X. Zhou, G. Li, C. Chai, and J. Feng, "A learned query rewrite system using monte carlo tree search," *Proc. VLDB Endow.*, vol. 15, no. 1, pp. 46–58, 2021.
[39] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR (Poster)*, 2015.
[40] B. Biggio, I. Corona, B. Nelson, B. I. P. Rubinstein, D. Maiorca, G. Fumera, G. Giacinto, and F. Roli, "Security evaluation of support vector machines in adversarial environments," *arXiv Preprint*, 2014. [Online]. Available: https://arxiv.org/pdf/1401.7727.pdf
[41] E. M. Kornaropoulos, S. Ren, and R. Tamassia, "The price of tailoring the index to your data: Poisoning attacks on learned index structures," in *SIGMOD Conference*, 2022, pp. 1331–1344.
[42] B. Ding, S. Das, R. Marcus, W. Wu, S. Chaudhuri, and V. R. Narasayya, "AI meets AI: leveraging query executions to improve index recommendations," in *SIGMOD Conference*, 2019, pp. 1241–1258.
[43] D. D. Chamberlin, *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann, 1998.
[44] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," in *SIGMOD Conference*, 1979, pp. 23–34.
[45] P. E. O'Neil, E. J. O'Neil, X. Chen, and S. Revilak, "The star schema benchmark and augmented fact table indexing," in *TPCTC*, ser. Lecture Notes in Computer Science, vol. 5895, 2009, pp. 237–252.
[46] R. C. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, "Neo: A learned query optimizer," *Proc. VLDB Endow.*, vol. 12, no. 11, pp. 1705–1718, 2019.
[47] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, "Bao: Making learned query optimization practical," in *SIGMOD Conference*, 2021, pp. 1275–1288.
[48] Z. Yang, A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica, "Neurocard: One cardinality estimator for all tables," *Proc. VLDB Endow.*, vol. 14, no. 1, pp. 61–73, 2020.
[49] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv Preprint*, 2014. [Online]. Available: https://arxiv.org/abs/1412.3555
[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.
[51] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.
[52] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *EMNLP*, 2015, pp. 1412–1421.
[53] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NIPS*, 2017, pp. 3146–3154.
[54] T. Siddiqui, W. Wu, V. R. Narasayya, and S. Chaudhuri, "DISTILL: low-overhead data-driven techniques for filtering and costing indexes for scalable index tuning," *Proc. VLDB Endow.*, vol. 15, no. 10, pp. 2019–2031, 2022.
[55] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel, "Self-critical sequence training for image captioning," in *CVPR*, 2017, pp. 1179–1195.
[56] Y. Yu, "Towards sample efficient reinforcement learning," in *IJCAI*, 2018, pp. 5739–5743.
[57] M. Khan, P. Srivatsa, A. Rane, S. Chenniappa, R. Anand, S. Ozair, and P. Maes, "Pretrained encoders are all you need," *arXiv Preprint*, vol. https://arxiv.org/abs/2106.05139, 2021.
[58] H. Kim, B. So, W. Han, and H. Lee, "Natural language to SQL: where are we today?" *Proc. VLDB Endow.*, vol. 13, no. 10, pp. 1737–1750, 2020.
[59] F. L. Deremer, "Generating parsers for BNF grammars," in *AFIPS Spring Joint Computing Conference*, ser. AFIPS Conference Proceedings, vol. 34. AFIPS Press, 1969, pp. 793–799.
[60] W. Wu, C. Wang, T. Siddiqui, J. Wang, V. R. Narasayya, S. Chaudhuri, and P. A. Bernstein, "Budget-aware index tuning with reinforcement learning," in *SIGMOD Conference*, 2022, pp. 1528–1541.
[61] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. R. Narasayya,

and S. Chaudhuri, "Selectivity estimation for range predicates using lightweight models," *Proc. VLDB Endow.*, vol. 12, no. 9, pp. 1044–1057, 2019.

[62] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *arXiv Preprint*, vol. https://arxiv.org/abs/2111.01243, 2021.

[63] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45.

[64] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT (1)*, 2019, pp. 4171–4186.

[65] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *ACL*, 2020, pp. 7871–7880.

[66] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," in *EMNLP (Findings)*, ser. Findings of ACL, vol. EMNLP 2020, 2020, pp. 1536–1547.

[67] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. M. V, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Moustafa-Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries, "Starcoder: may the source be with you!" *arXiv Preprint*, vol. https://arxiv.org/abs/2305.06161, 2023.

[68] E. Parisotto, H. F. Song, J. W. Rae, R. Pascanu, Ç. Gülçehre, S. M. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, M. M. Botvinick, N. Heess, and R. Hadsell, "Stabilizing transformers for reinforcement learning," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 119, 2020, pp. 7487–7498.

[69] W. Li, H. Luo, Z. Lin, C. Zhang, Z. Lu, and D. Ye, "A survey on transformers in reinforcement learning," *arXiv Preprint*, vol. https://arxiv.org/abs/2301.03044, 2023.

[70] M. Andrychowicz, A. Raichuk, P. Stanczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters for on-policy deep actor-critic methods? A large-scale study," in *ICLR*, 2021.

[71] X. Guo and H. Yu, "On the domain adaptation and generalization of pretrained language models: A survey," *arXiv Preprint*, vol. https://arxiv.org/abs/2211.03154, 2022.

[72] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *arXiv Preprint*, 2020. [Online]. Available: https://arxiv.org/abs/2006.14171

[73] K. Schnaitter, N. Polyzotis, and L. Getoor, "Index interactions in physical design tuning: Modeling, analysis, and applications," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 1234–1245, 2009.

[74] Y. Luo, J. Peng, and J. Ma, "When causal inference meets deep learning," *Nat. Mach. Intell.*, vol. 2, no. 8, pp. 426–427, 2020.

[75] D. Kalainathan, O. Goudet, and R. Dutta, "Causal discovery toolbox: Uncovering causal relationships in python," *J. Mach. Learn. Res.*, vol. 21, pp. 37:1–37:5, 2020.

[76] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[77] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, 2001.

[78] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 1, pp. 3:1–3:39, 2012.

[79] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *SIGMOD Conference*, 2000, pp. 93–104.