Leveraging Dynamic and Heterogeneous Workload Knowledge to Boost the Performance of Index Advisors

Zijia Wang
Haoran Liu
Chen Lin
School of Informatics, Xiamen University
China
chenlin@xmu.edu.cn
zjwang@stu.xmu.edu.cn
liuhr@stu.xmu.edu.cn

Zhifeng Bao
RMIT University
Australia
zhifeng.bao@rmit.edu.au

Guoliang Li
Department of Computer Science, Tsinghua University
China
liguoliang@tsinghua.edu.cn

Tianqing Wang
Huawei Company
China
wangtianqing2@huawei.com

ABSTRACT

Current index advisors often struggle to balance efficiency and effectiveness when dealing with workload shifts. This arises from ignorance of the continual similarity and distant variety in workloads. This paper proposes a novel learning-based index advisor called BALANCE, which boosts indexing performance by leveraging knowledge obtained from dynamic and heterogeneous workloads. Our approach consists of three components. First, we build separate Lightweight Index Advisors (LIAs) on sequential chunks of similar workloads, where each LIA is trained with a small batch of workloads drawn from the chunk, and it provides direct index recommendations for all workloads in the same chunk. Second, we perform a policy transfer mechanism by adapting the LIA's index selection strategy from historical knowledge, substantially reducing the training overhead. Third, we employ a self-supervised contrastive learning method to provide an off-the-shelf workload representation, enabling the LIA to generate more accurate index recommendations. Extensive experiments across various benchmarks demonstrate that BALANCE improves the state-of-the-art learning-based index advisor, SWIRL, by 10.03% while reducing training overhead by 35.70% on average.

PVLDB Reference Format:
Zijia Wang, Haoran Liu, Chen Lin, Zhifeng Bao, Guoliang Li, and Tianqing Wang. Leveraging Dynamic and Heterogeneous Workload Knowledge to Boost the Performance of Index Advisors. PVLDB, 17(7): 1642 - 1654, 2024.
doi:10.14778/3654621.3654631

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

1 INTRODUCTION

Selecting appropriate attributes to build indexes can accelerate data retrieval at the cost of storage overhead and index maintenance [27]. This can be formalized as the Index Selection Problem (ISP), which is to select the set of indexes for a given workload to maximize the workload's performance while satisfying the storage cost constraints.

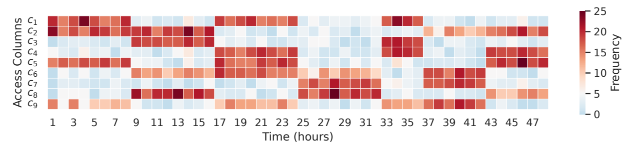


Figure 1: An illustrative example of the frequency of columns accessed by the workload every hour

The ISP has been proven to be NP-hard [28], and many index advisors have been proposed to solve it. Early Index Advisors (IAs) are usually heuristic-based [3, 4, 6, 12, 32, 36, 37], which use predefined rules to add or remove indexes to the output index configuration iteratively. Recently, learning-based IAs have emerged, and they are mostly based on deep Reinforcement Learning (RL) [17, 18, 22, 31] or Monte Carlo tree search [38, 41] to make index selection decisions.

However, in real-world scenarios, workloads evolve over time. Existing IAs struggle to deliver effective index configurations efficiently for dynamic and heterogeneous workloads due to the costly tuning overhead (e.g., time cost to train IAs, time cost to determine index selections, etc.). We argue that they neglect two important temporal properties of workloads, i.e., *continual similarity* and *distant variety*.

The *continual similarity* refers to situations where the workload experiences relatively minor changes in the short term. As shown in Figure 1, workloads from hour 1 to hour 8 have similar query patterns, e.g., access columns $c1, c2, c5$ with the same frequency. This property has been observed in many production database systems. For example, BusTracker [1] is a mobile phone application for live-tracking of the public transit bus system. On weekdays during commuting hours, the workloads contain many commute-related queries, such as checking bus arrival times, planning routes to desired destinations, etc. Another example is App-X [34] for trading on the stock exchange. During the opening bell, most queries are value-based, such as stock buyers searching for the current price of specific stocks or checking market trends.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 17, No. 7 ISSN 2150-8097.
doi:10.14778/3654621.3654631

[1] http://www.cs.cmu.edu/~malin199/data/tiramisu-sample

The *distant variety* pertains to situations where the workload undergoes significant evolution over the long term. As shown in Figure 1, workloads from hour 9 to hour 16 access columns $c2$, $c3$, $c6$, $c8$, which is a different set of columns than hour $1-8$. Distant variety is also observed in real scenarios, and it can result from a shift in user intent or workload evolution. For example, Bustracker queries during weekends are more activity-related, such as users exploring bus schedules and alternative routes for leisure activities. In the case of App-X [34], queries during the market closing time are more technology-based, such as the securities company staff performing transaction monitoring of specific stock categories. Moreover, workload shift is observed [23] in MOOC [2] whenever new features are released because more queries exploring and utilizing new functionalities emerge.

The *continual similarity* and *distant variety* raise two questions for learning-based index advisors.

First, existing learning-based index advisors [17, 18, 31] often overlook continual similarity and need to implement expensive trials to fit each workload. A natural question arises: *can we train the IA with a small batch of samples and use it to directly predict indexes for similar workloads in a certain period of time?* Since workloads in adjacent hours have common query patterns and demands, they would benefit from employing similar index selection strategies. Here, we emphasize that directly recommending indexes for similar workloads is non-trivial because the indexing policy is based on accurate workload representation that captures the subtle, index-aware differences in similar workloads.

Second, when workloads evolve, existing learning-based index advisors utilize only the index trials on the current workload and discard potentially valuable historical index selection experiences on distant workloads in the past. *Can we improve the learning efficiency and reduce the training overhead of index advisors by fully exploiting historical samples on past workloads?* Since machine learning models can improve themselves through training experiences, it is plausible that with appropriate treatments, past experiences can give the index advisor a better initialization on the current workload, and the index advisor will converge to the optimal index selection faster. Nonetheless, integrating historical experiences into the current training process is challenging because of the distant variety. Clearly, the index selection strategies need to adapt and undergo considerable changes to cater to the varied requirements of distant workloads. The indexing policy must be carefully designed to transfer knowledge from past experiences without negatively impacting the current workload.

Most RL-based index advisors take a long training time to achieve sufficiently good index recommendations, primarily because they overlook the importance of continual similarity and distant variety in their training process. For example, SWIRL [17] demonstrates the state-of-the-art effectiveness for dynamic workloads. Yet, it still requires approximately of training duration on the TPC-H 10GB dataset. By effectively leveraging the continual similarity and distant variety, as we will show shortly, we only need 3.3 min of training duration to achieve the same indexing performance.

[2]https://www.mooc.org/

Our Contributions. To address the above issues, we propose BALANCE: a transfer RL-based index advisor for dynamic workloads in real scenarios.

First, BALANCE builds separate Lightweight Index Advisors (LIAs) on sequential chunks of similar workloads. Each LIA is trained with a small batch of samples drawn from the chunk, instead of the whole chunk (i.e., lightweight). It can provide direct index recommendations for other workloads in the same chunk (Section 3.2). BALANCE achieves comparable indexing results to the near-optimal Extend [32] with less than 0.6% inference runtime.

Second, BALANCE presents a policy transfer mechanism to adapt the current LIA from previously trained LIAs based on workload similarities. Thus, knowledge learned from distant workloads is transferred without introducing noise or fusing dissimilar workloads, and the training efficiency of reinforcement learning is improved (Section 5). BALANCE improves SWIRL by 10.03% while reducing the training overhead by 35.70% on average.

Third, BALANCE employs a self-supervised contrastive learning method before training LIAs to obtain off-the-shelf workload representations. Thus, the workload representations are obtained more efficiently without actually implementing the time-costly reinforcement learning trials. Furthermore, the workload representations reveal key characteristics of the workloads related to indexing performance and enable LIAs to produce more reliable and accurate index recommendations (Section 4). Ablation study on BALANCE shows that, compared with workload representations extracted from query text [31], the self-supervised workload representation can reduce workload execution cost by up to 6.6%.

2 RELATED WORK

2.1 Index Advisor

Some heuristic-based index advisors reduce a comprehensive set of initial index candidates step by step [3, 37]. These methods often lead to excessively long runtime because many iterations are required to satisfy the specified budget [16]. Other works add indexes iteratively to an empty set, where the indexes can be single-column indexes [12] or multi-column indexes [4, 6, 32, 36].

Recently, learning-based methods based on reinforcement learning [35] have shown great potential in both efficiency and accuracy. Their difference mainly lies in *state representations*, which can be (1) workload-independent [27, 41], e.g., treating potential index combinations as tree nodes and fetching index configurations via Monte Carlo Tree Search (MCTS); (2) query level [18], e.g., including the frequency of queries; (3) column level [30, 31], e.g., incorporating a selectivity vector of each attribute; or (4) plan level [17], e.g., incorporating query operators parsed from the execution plan.

As per [16], heuristic methods are shown to work poorly for large databases and complex workloads because such methods cannot balance between high inference efficiency and high index quality. As per [17], Extend [32] produces the best workload execution cost, and SWIRL [17] makes comparable workload cost reduction while significantly reducing inference time and training overhead.

2.2 Transfer Reinforcement Learning

Reinforcement learning faces the problem of sparse feedback and sample inefficiency, especially in high-complexity state and action spaces, where obtaining enough interaction samples is prohibitive [1]. Transfer learning has emerged as a promising approach to address the problem of sample inefficiency and accelerate RL [42].

One line of work transfers reward/value function, such as using reward shaping [5] or estimating the reward function from source task samples [19]. Nevertheless, the transfer of value functions or samples typically depends on accurate estimations or prior knowledge for measuring similarity. This reliance increases computational complexity, making it infeasible for practical applications.

Another line of work tries to reuse the policy directly in the source tasks. This method usually considers many-to-one scenarios. For instance, probabilistic policy reuse assigns probabilities to each source policy based on their expected performance gain in the target domain [11], generalized policy improvement extends one source policy to multiple policies [2], optimize source policy selection using MAB techniques [21], etc.

Remarks. (1) Existing reinforcement-learning-based index advisors ignore the historical workloads and hurt their training overhead and index performance. Our proposed scheme introduces separate lightweight index advisors with a policy transfer mechanism. This approach enhances the efficiency of reinforcement learning by transferring knowledge from distant workloads while maintaining strong generalization capabilities within each index advisor. (2) Workload representation is vital in ISP, and most previous studies rely on fixed strategies to extract workload representations from queries [17, 31]. We argue that they do not accurately reflect the characteristics between workloads and their impacts on index selection. Thus, we present a self-supervised contrastive learning method to obtain workload representations relevant to index selection.

3 SOLUTION OVERVIEW

In this section, we start with preliminaries, followed by a formal definition of the index selection problem, as well as an overview of our solution. Frequently used notations are listed in Table 1.

3.1 Problem Formulation

A workload refers to a set of queries with their frequencies, i.e., $w = (q_1, f_1), \cdots, (q_{|w|}, f_{|w|})$, where q_i is a query, f_i is the frequency of q_i, $|w|$ is the number of queries in workload w.

Definition 3.1. Index Selection Problem (ISP). For a certain database $\mathcal{D}$, given a workload w, a set of index candidates $\mathbb{I}$, and a storage budget B, the ISP is to determine an index configuration $I^* \subseteq \mathbb{I}$ so that $I^* = argmin_{I \subseteq \mathbb{I}} cost(w, I)$ subject to $M(I) \leq B$.

Here the workload cost under an index configuration I^* is an aggregation of each query cost in the workload, $cost(w, I) = \sum_{i=1}^{|w|} f_i \cdot cost(q_i, I)$, where $cost(q_i, I)$ is the execution cost of q_i which is evaluated by a what-if caller [7]. The total storage of an index candidate amounts to $M(I) = \sum_{i \in I} m_i$, where m_i is the required storage of index i.

3.2 Overview of Our Solution

We first explain the motivation. Workloads for real-world applications are never static [23]. Among the few existing IAs that address dynamic workloads, they struggle to balance training efficiency

Table 1: Frequently used notations

Notations	Brief Description
W	A workload chunk contains a set of similar workloads
π	Agent policy of a sub-model for a workload chunk
w	Workload contains a set of queries with their frequencies
$\mathbb{I}$	A set of index candidates for a workload
I^*	Index configuration selected from index candidates
$cost(w, I)$	Execution cost of workload w under index configuration I
$M(I)$	Required storage of index configuration I
B	Storage budget of index configuration
d_{max}	The maximum number of attributes contained in an index
$\mathbf{p}$	The off-the-shelf representation of workload

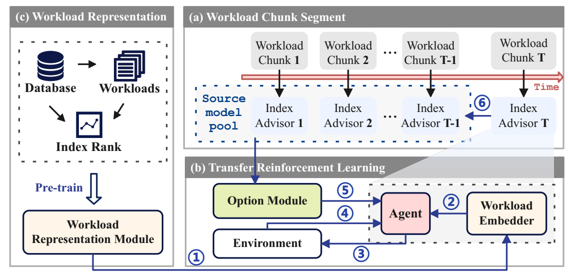


Figure 2: Framework overview of BALANCE

and effectiveness. For example, they either repetitively run expensive indexing trials on the workload at hand and deliver the best index configuration during the trials [31], or train one model on the entire set of workloads and their variants with extensive training overhead to enable direct index recommendation [17].

Our goal is to directly make accurate index selections without training on a large set of workloads. To achieve this goal, the key idea is to split the workloads into chunks of similar workloads, build a separate IA for each chunk, train it with a small portion of workloads sampled from the chunk, and use it to make direct index selections for other workloads in the chunk. Such a framework is more efficient than existing index advisors because (1) each separate IA is lightweight, i.e., training with a small number of workloads requires significantly fewer interactions with the database environment and leads to less training time and fewer resources, and (2) each index advisor is in a one-off fashion, i.e., once the model is trained, it can directly recommend index selections for any workload and no trial indexes are required.

We also want the IAs to learn from past experiences enhancing their robustness to workload shifts. Since the workloads are dynamic and heterogeneous, a policy transfer mechanism is required to transfer useful knowledge (e.g., learned from similar historical workloads instead of simply mixing all indexing experiences).

To improve index accuracy, it is necessary to capture workload characteristics and encode them as representation vectors. A better workload representation can (1) facilitate the index advisor to adapt its index selection strategy to different workloads; (2) identify workloads with potential relevant index performances and transfer knowledge learned from the appropriate workloads. Most prior index advisors extract the workload representations from query text and query plans via predefined strategies, which is sub-optimal. For example, extracting operators in query plans and neglecting the range values [17]. Because the information within the query text or plan typically does not directly relate to index selectivity or does not cover the details of index selection comprehensively, these approaches often fail to accurately capture the relationship between the query and index selection. Thus, the workload representations must be learned to distinguish subtle differences in workloads that are crucial for index selections.

Ideally, the workload representation learning should be off-the-shelf, i.e., the workload representations should be obtained before the IA's training procedure because updating the workload representation during the index learning process can be time-consuming and impractical. Secondly, the parameters to encode workload representations should not be updated during the index learning procedure. Otherwise, since only a limited number of queries and indexes are exposed in producing indexing trials, the workload representations learned would be less discriminative.

To achieve the above, we propose BALANCE (**B**oosting index **A**dvisor by **L**ever**A**ging dy**N**ami**C** and Heterog**E**neous workload). As shown in Figure 2, BALANCE contains three major components, and they are implemented in the following workflow.

Before the index recommendation process, BALANCE proposes an offline workload representation module (Section 4). This module outputs a numerical vector (i.e., representation) for each input workload. The module adopts self-supervised contrastive learning, i.e., the training of workload representation does not involve an index advisor. It automatically generates a set of workloads, implements them on a database given certain heuristically defined indexes, observes their index preferences (i.e., Index Rank), and optimizes a neural network to obtain the workload representation.

In a nutshell, the workload stream is segmented into chunks of similar workloads determined by the differences between their workload templates (more details in Section 6.7), and a separate index advisor is built for each chunk. Each index advisor is an agent trained by transfer RL.For a new workload chunk W^T currently arrived, a small portion of workloads is sampled, and BALANCE implements the following steps. ① Call the off-the-shelf workload representation module and initialize the workload representation. ② The workload representation, the current index configuration, and other meta information are sent to the agent as the current state. ③ The agent generates an action, i.e., choosing an index based on the current state, according to its action policy. ④ The environment provides the reward of the current action via the what-if optimizer to the agent. ⑤ The option module decides whether and how to update the current policy by transferring knowledge from previous policies, i.e., index advisors for chunk $1, \cdots, T-1$ [3]. The process iterates from step ② until the maximum number of iterations or the indexing budget is exceeded. After the index advisor T is trained, we use it to offer index recommendations for other workloads in chunk T. ⑥ We put the current index advisor T into the source model pool and proceed to chunk $T+1$.

[3] In practice, for a large and complex dataset, we can keep at most k most recent index advisors $T-k+1, \cdots, T$ in the source model pool (i.e., keep a fixed number of source IAs). An experimental study on the number of k is shown in Section 6.6.

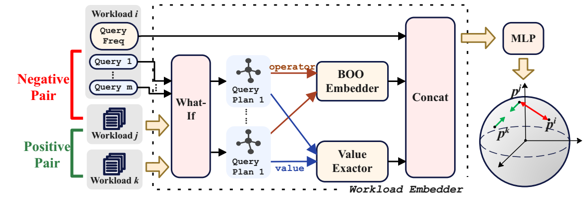


Figure 3: Workload representation module architecture

4 WORKLOAD REPRESENTATION LEARNING

For each input workload w, the workload representation module aims to output a numerical vector $\mathbf{p}^w \in \mathbb{R}^{D^p}$ called the workload representation, where D^p is the embedding size.

4.1 Architecture

We focus on learning to encode $\mathbf{p}$ instead of defining heuristic strategies [17, 18, 31], because pre-fixed strategies are incomplete to derive a fine-grained understanding of the workload characteristics to guide index selection. Moreover, the workload representation module is trained off-the-shelf, meaning it is trained prior to the index recommendation process without actually running the workloads in the system to avoid resource consumption in the production workflow.

We resort to self-supervised contrastive learning [15], which has demonstrated superior performance in computer vision [13] and natural language processing [24] due to its capability of learning from unlabeled data. Our objective is to unveil the workload's hidden characteristics without the index labels because it is infeasible to gather the gold standard labels, i.e., the Index Selection Problem is NP-hard, and we can only obtain near-optimal index labels at best. Consequently, contrastive learning is suitable and scalable for our problem.

As shown in Figure 3, first, we randomly generate a set of workloads. Then, for each batch of workloads, we call the what-if optimizer to extract features from the query plans and concatenate these features. The features go through a network to obtain representation vectors for each workload. Finally, we optimize the model parameters based on the contrastive loss, i.e., bringing positive sample pairs closer together and pushing negative sample pairs farther apart in the representation space.

4.2 Constructing positive and negative pairs

Contrastive learning extracts meaningful features by discerning similarities and differences between samples, aiming to bring similar samples closer while pushing dissimilar ones apart. Constructing positive and negative samples is crucial. For example, in computer vision, the original image is treated as an anchor, and the positive samples are constructed by image augmentation [8] (color jittering, random cropping, and resizing, etc.), while other samples are used as negative samples. In natural language processing, the positive sample can be obtained by replacing or masking random tokens in the sequence [40].

None of these augmentation techniques are feasible for our problem. For example, dropping a sub-clause will largely affect the index selection of a query; masking several tokens will make the query non-executable. Thus, the augmented queries will not be considered similar to the anchor query.

We propose an automated procedure to construct samples and identify positive and negative pairs. Suppose we generate a set of workloads $\mathcal{N} = \{w\}$ [4]. For each workload w, we extract the relevant columns and derive a set of index candidates $\mathbb{I}^w$ where the maximal index width is d_{max}, which is a pre-defined hyper-parameter.

Next, we measure the similarity between workloads by their index performances. We propose *Index Rank* to reflect a workload's index preference. The what-if optimizer is invoked to assess and rank the index candidates based on the estimated cost of the workload w.r.t. each index candidate. Formally, for every workload w, an index rank sequence R^w is defined as:

$$R^w =< I_1^w, I_2^w, \cdots, I_{n^w}^w >,$$
$$\forall x < y, cost(w, I_x^w) < cost(w, I_y^w), \tag{1}$$

where $I_i^w \in \mathbb{I}^w, \forall 1 \leq i \leq n^w$, and n^w is the size of $\mathbb{I}^w$. Note that Index Rank is independent of index advisors, meaning that we can perceive the workload's index preference without specifying an index advisor.

Then, we present a set-based correlation metric to quantify the similarity between two workloads:

$$\alpha(w, w') = \sum_{t=1}^{\min_{i \in \mathcal{N}} n^i} \frac{|R_{1:t}^w \cap R_{1:t}^{w'}|}{t}, \tag{2}$$

where $R_{1:t}^w$ denotes the set of all elements in the R^w from position 1 to position t. The correlation measures from position $t = 1$ to the minimal length of all index ranks in the training set, calculates the set similarity between two workloads at each position t of their index ranks, and aggregates the similarity at all positions.

Lastly, we compute the similarity distribution among all pairs (i.e., $|\mathcal{N}| \times |\mathcal{N}|$, $|\mathcal{N}|$ is the number of workloads in $\mathcal{N}$) of workloads within the training set. For any anchor workload w, workloads with a similarity value $\alpha(w, w') > \tau^+$ is its positive sample set $e^+(w)$, and others are the negative set $e^-(w)$, where τ^+ is the threshold.

As illustrated in Figure 4, we derive the index candidates of workloads w_1, w_2, w_3 and obtain the index rank sequences via what-if optimizer. Based on these sequences, we calculate the similarity between them to determine positive samples (e.g., $< w_1, w_2 >$) and negative samples (e.g., $< w_1, w_3 >$ and $< w_2, w_3 >$).

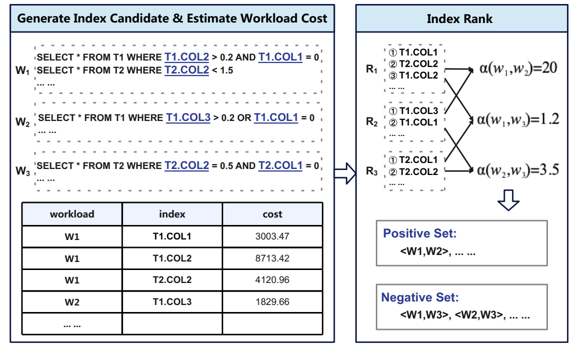


Figure 4: Example of constructing positive and negative pairs

4.3 Workload Feature

We encode the semantic information of workloads. We build tree-structured query plans by utilizing the DBMS's optimizer (i.e., what-if optimizer) and Bag Of Operator (BOO) to encode the query operators. Specifically, the operators of each plan relevant to index selection are converted into a textual representation. For example, in the filter condition `T1.COL2<5`, a textual representation `TabScan_T1_COL2_Pred<` will be generated.

An operator dictionary is constructed to store all textual representations. Suppose the size of the operator dictionary is K, for each query i, the BOO embedder uses a sequence of binary indicators $b_i =< b_{i,1}, \cdots, b_{i,K} >$, where $b_{i,k} = 1$ suggests that the query contains the $k-$th operator in the operator dictionary. Since the features produced by BOO embedder are too sparse, we construct a Latent Semantic Indexing (LSI) model [9] for dimension reduction, and the output is denoted as $\mathbf{l}_i$.

[4] The workloads can be populated from pre-defined query templates.

Next, we proceed to encode the value of the query. Similar to operator encoding, we parse the query plan and extract the values corresponding to the operator. For example, in the filter condition `T1.COL2<5`, there is the value `5` corresponding to the operator `TabScan_T1_COL2_Pred<`. We maintain a sequence of value encoding, $v_i =< v_{i,1}, \cdots, v_{i,K} >$, where $v_{i,k}$ is the value representation corresponding to the $k-$th operator in the operator dictionary. Values can be divided into two types, including the numeric value and string value:

(1) For `numeric value`, we normalize it, i.e., $v_{i,k} = \frac{val-min}{max-min}$, where val is the value in the query, min is the minimum value, and max is the maximum value of the respective column in the training data.
(2) For `string value`, we encode it with a data binning representation. For a column of string type, its H-size value binning borderline is denoted as $u =< u_0, \cdots, u_H >$, where u_h suggests that in this column h/H of the values are smaller than u_h in lexicographic order. In this way, we locate the bin $h : u_{h-1} < val \leq u_h$, and the string value is encoded as $v_{i,k} = \frac{h}{H}$.

For $v_{i,k}$ where the k-th operator does not correspond to any values, padding can be applied, i.e., $v_{i,k} = 0$. As operator encoding, we use PCA to reduce the dimension of v_i. Finally, the value encoding is denoted as $\mathbf{c}_i$.

We concatenate operator encoding, value encoding, and query frequency of each query in a workload,

$$\mathbf{z}^w = [\mathbf{l}_i, \mathbf{c}_i, f_i]_{i=1}^{|w|}. \tag{3}$$

4.4 Training

The workload feature vector $\mathbf{z}^w$ is fed into a Multi-Layer Perceptron (MLP) to obtain the workload representation:

$$\mathbf{p}^w = MLP(\mathbf{z}^w). \tag{4}$$

The parameters of the MLP are optimized via the contrastive loss, i.e., minimizing the distance between positive pairs in e^+ and maximizing the distance between negative pairs in e^-:

$$\mathbb{L}_C = -log \sum_{w \in \mathbb{B}} \frac{\sum_{w' \in e^+(w)} \mathbf{p}^w \cdot \mathbf{p}^{w'}}{\sum_{w' \in e^-(w)} \mathbf{p}^w \cdot \mathbf{p}^{w'}}, \tag{5}$$

where $\mathbb{B}$ is a batch of training instances.

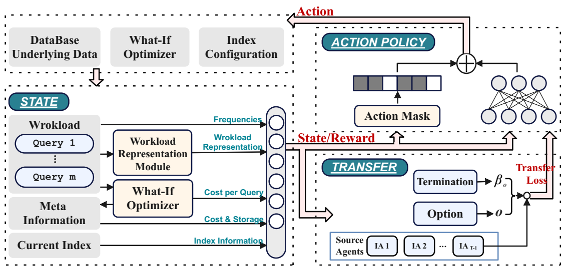


Figure 5: Architecture of the index advisor

5 TRANSFER RL INDEX ADVISOR

Let $W^t, t = 1, \cdots, T$ denote the workload chunk t, and T is the maximal number of chunks. Each workload chunk contains a set of workloads, i.e., $W^t = \{w_1^t, \cdots, w_{E^t}^t\}$, where E^t is the number of workloads in chunk t. For each workload chunk, a lightweight index advisor is trained on the training workloads, $\pi = \{\pi_1, \cdots, \pi_T\}$.

The backbone of each index advisor is based on SWIRL [17], which is a reinforcement learning method. In RL, the IA acts as an agent and interacts with the DB environment. During training, a portion of workloads is sampled from the workload chunk as training workloads. Let's denote them as $\tilde{W}^t = \{w_1^t, \cdots, w_{V^t}^t\}$, where $V^t \ll E^t$. An IA repeatedly produces index configurations on the training workloads. Specifically, at each step i, the agent observes the environment and encodes the current state in a vector $\mathbf{s}_i$. Based on the current state $\mathbf{s}_i$, the agent selects an action a_i, which is an index to be added to the index configuration I^*, and the procedure repeats until the index budget is satisfied, i.e., $M(I^*) \leq B$.

BALANCE improves SWIRL on two major aspects. (1) BALANCE provides a more accurate state by integrating the fine-grained workload representation in Section 4. (2) BALANCE enhances each index advisor by transferring knowledge from previous index advisors. As shown in Figure 5, BALANCE consists of three components, `state representation`, `action policy`, and `transfer network`.

5.1 State Representation

In reinforcement learning, state representation plays a crucial role as it compresses the complex, high-dimensional environmental input into a low-dimensional vector that reveals key attributes for the task at hand. Our approach employs a comprehensive state representation, as depicted in Figure 5, which includes the workload information, meta information, and index information.

Workload information. First, we encode the query semantics to offer the agent a good understanding of the workload. Thus, we feed the workload $w = (q_1, f_1), \cdots, (q_{|w|}, f_{|w|})$ into the workload representation model and obtain $\mathbf{p}^w$. To reflect the performance of the current action, we define $\mathbf{s}^F \in \mathbb{R}^{|w|}$, where $s_j^F = f_j$ is the frequency of each query j, and $\mathbf{s}^E \in \mathbb{R}^{|w|}$, where $s_j^E = cost(q_j, a_{<i})$ is the estimated cost of each query under the current index configuration. The workload information is a vector $\mathbf{s}^W \in \mathbb{R}^{D^W} = [\mathbf{p}^w, \mathbf{s}^F, \mathbf{s}^E]$, where the embedding size is pre-determined.

Meta information. Meta information is a fixed-length vector $\mathbf{s}^M \in \mathbb{R}^4$, containing four scalar features: (1) the specified storage budget, (2) the current storage consumption, (3) the workload execution cost without any indexes, and (4) the workload cost under the current index configuration estimated by what-if optimizer.

Index information. The index information is a vector $\mathbf{s}^I \in \mathbb{R}^{D^I}$ that encodes the current action for each indexable column, with D^I being the number of indexable columns. For multi-column indexes, since the primary column is more important, the value of each column is decremented based on its position in the index. Formally, $\mathbf{s}_j^I = \sum_{a_i} 1/k(j, a_i)$ where $k(j, a_i)$ is the position of column j in action a_i.

Finally, the current state concatenates the above information vectors, i.e., $\mathbf{s} = [\mathbf{s}^W, \mathbf{s}^M, \mathbf{s}^I]$.

5.2 Action policy

The action policy component is a 2-layer Feed-Forward Network (FFN) taking the state $\mathbf{s}_i$ as input. The index advisor selects an action a_i based on the value of $\pi_t(a_i, \mathbf{s}_i)$. Note that we use discrete actions; each action corresponds to choosing an index from the index candidates. For each workload, we generate all syntactically relevant index candidates according to the specified maximal index width d_{max}. For instance, for query `"select * from T1 where T1.COL1>3 and T1.COL3<5"` we generate index candidates { `(T1.COL1),(T1.COL3), (T1.COL1,T1.COL3), (T1.COL3,T1.COL1)` }.

The complexity of index selection is closely related to the index candidate space. Too many actions can affect the convergence speed of RL algorithms. Additionally, when we consider specific workloads, particular actions might be illegal (e.g., indexes that exceed the storage budget).

To address the issues above, we employ action masking before each step i to control the action space. (1) `Pruning indexes that exceed the storage budget.` Before each step, we consider the current storage consumption and ensure the next actions will not exceed the specified budget. (2) `Deleting indexes with an invalid precondition.` According to the intuition in [6] "that for a two-column index to be desirable, a single-column index on its leading column must also be desirable", at each step i, we mask a multi-attribute index if its primary column is not selected in previous actions $a_{<i}$.

5.3 Transfer network

Conventional RL training, i.e., training the lightweight index advisor from scratch on the training workload, takes a large training overhead. Because RL training needs to obtain feedback from the environment by running (or estimating) the workload on different index actions, it converges slowly. Furthermore, because of the limited samples in the training workloads and the enormous amount of possible workloads in the workload size, conventional RL training can not generalize well to dynamic workloads.

To address the above issues, we incorporate a transfer network component in the IA.Transfer learning exploits the knowledge gained from a source task or domain to improve generalization on a target task or domain. Our idea is to transfer the knowledge (i.e., action policy) of appropriate agents well-trained in the past to guide the learning of the current agent. In this manner, the convergence will be more speedy with the supervision of more ripe agents, and the learned policy will be more robust to dynamic workloads because the policy is trained with a larger sample complexity.

Note that we do not simply reuse a previous agent since we want to use all training workloads fully. Instead, we use previous agents to regulate the action of the current agent. Also, we do not fix a source supervisor, i.e., the appropriate agents are different in each state. Allowing the supervisor to change according to the state can provide more accurate supervision.

Inspired by [39], the transfer network is decomposed into two sub-modules, i.e., the termination network and the option network. Due to the same input and the relatively simple functionality of the termination and option networks, to increase the inference efficiency, we employ 2-layer FNNs. First, we initialize a set of options denoted by $O = \{\mathbf{o}_1, \cdots, \mathbf{o}_{T-1}\}$, where $\mathbf{o}_j \in \mathbb{R}^{T-1}$ is a one-hot vector. The j-th element in $\mathbf{o}_j$ is set to 1, representing the previous agent π_j, i.e., $\pi_{\mathbf{o}_j} = \pi_j$. The termination network takes a state and an option as input and outputs the probability of changing the current supervisor $\beta(\mathbf{s}, \mathbf{o}) \in [0, 1]$. The option network also takes the state and an option as input and outputs the probability of taking the particular option as the supervisor $Q(\mathbf{s}, \mathbf{o}) \in [0, 1]$.

Suppose at workload chunk T, the index advisor π_T can seek guidance from previous agents $\pi_1, \cdots, \pi_{T-1}$. At the first step of a training trial (i.e., producing the index configuration for a training workload), the transfer network selects an option $\mathbf{o}$ by the option network, i.e., $\mathbf{o} = argmax_{\mathbf{o} \in O} Q(\mathbf{s}, \mathbf{o})$. In the following steps, the index advisor will judge whether or not to keep choosing the supervisor by drawing a switch from β. If the switch terminates the current choice, another supervisor will be chosen based on option network Q.

5.4 Update action policy

To update the action policy π, we employ proximal policy optimization (PPO) [33], because ISP involves a large number of indexes to choose from, and PPO can handle a complex action space and provide faster updates.

The action policy is updated to maximize the expected reward of all training trials. We utilize relative costs as the reward, which calculates the ratio of cost reduction concerning the initial cost. We also consider the impact on index storage and incorporate the storage increment before and after the index selection in the reward at each step r_i. This encourages the agent to consider storage consumption while striving to reduce workload costs.

$$r_i = \frac{cost\left(w, I^*_{i-1}\right) - cost\left(w, I^*_i\right)}{cost\left(w, I^*_0\right) * \left(M\left(w, I^*_i\right) - M\left(w, I^*_{i-1}\right)\right)}, \tag{6}$$

where $cost\left(w, I^*_{i-1}\right)$ and $M\left(w, I^*_{i-1}\right)$ are the workload cost and storage consumption of indexes made by previous $i-1$ action steps.

Furthermore, we consider the transfer loss. The supervisor at each state is also recorded, and the transfer loss is defined between the source policy and the current policy $\mathbb{L}_H = H(\pi_{\mathbf{o}} || \pi_T)$, where $H(\cdot||\cdot)$ is the cross-entropy loss. We combine the reward and the transfer loss, weighted by an attenuation factor δ_t, in the loss dependent on the action policy:

$$\begin{aligned}\mathbb{L}(\pi_T) = \sum_i \Big(& r_i \pi_T(a_i, \mathbf{s}_i) + \delta_t \big(\pi_T(a_i, \mathbf{s}_i) log \pi_{\mathbf{o}}(a_i, \mathbf{s}_i) \\ & + (1 - \pi_T(a_i, \mathbf{s}_i)) log(1 - \pi_{\mathbf{o}}(a_i, \mathbf{s}_i))\big)\Big).\end{aligned} \tag{7}$$

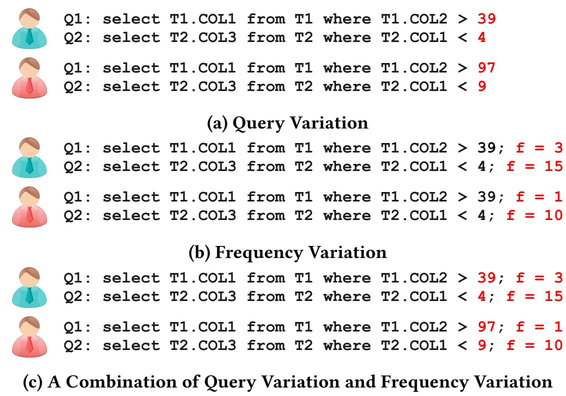


Figure 6: Example of three workload chunk settings

5.5 Update transfer network

We opt for a value-based approach [25] to update the transfer network because the number of source policies in the transfer network is significantly smaller than the number of index actions, and using value-based approaches is faster.

To update the option network $Q_{\mathbf{o}}(\mathbf{s}, \mathbf{o})$, we first sample a batch of D transitions $(\mathbf{s}_i, a_i, r_i, \mathbf{s}_{i+1}, \mathbf{o}_i)$ from replay buffer $\mathbb{D}$ that records the state, action, reward, next state, and the supervisor option at a step i. Then, we use the source policy $\pi_{\mathbf{o}}$ of option $\mathbf{o}_i$ to select an action $\hat{a}_i$ at state $\mathbf{s}_i$. If the source policy outputs the identical action with the target policy, i.e., $\hat{a}_i = a_i$, we can use the expected return of the transition to update the option network. The objective minimizes the difference between the option network and the expected return.

$$\mathbb{L}(Q_{\mathbf{o}}) = \frac{1}{D} \sum_i (g_i - Q_{\mathbf{o}}(\mathbf{s}_i, \mathbf{o}))^2, \tag{8}$$

where g_i is the expected return at step i and is generally defined as:

$$g_i = r_i + \gamma U(\mathbf{s}_{i+1}, \mathbf{o}), \tag{9}$$

the quality of the next state given source $\mathbf{o}$ is measured by $U(\mathbf{s}_{i+1}, \mathbf{o})$ and is combined to the expected return g_i with a discount factor γ.

Based on the transfer mechanism in Section 5.3, the quality of the next state given source $\mathbf{o}$ is defined as

$$U(\mathbf{s}_{i+1}, \mathbf{o}) = \beta(\mathbf{s}_{i+1}, o) \max_{\mathbf{o}' \in O} Q_{\mathbf{o}'}(\mathbf{s}_{i+1}, \mathbf{o}') + \big(1 - \beta(\mathbf{s}_{i+1}, \mathbf{o})\big) Q_{\mathbf{o}}(\mathbf{s}_{i+1}, \mathbf{o}), \tag{10}$$

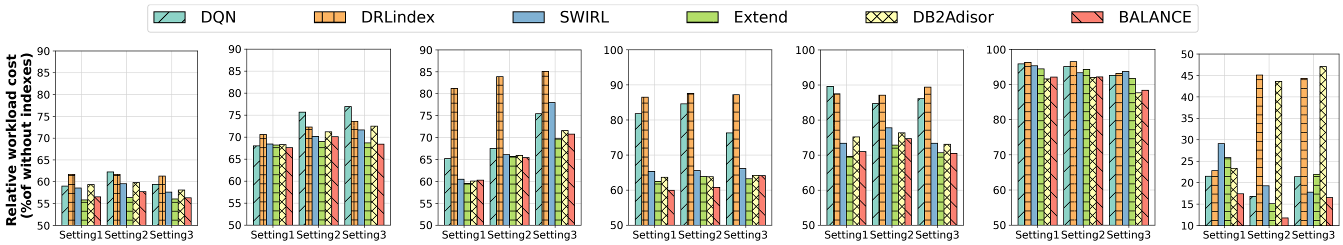
where $\beta(\mathbf{s}_{i+1}, \mathbf{o})$ is the terminate network which outputs the probability of determining the current option $\mathbf{o}$. If $\beta(\mathbf{s}_{i+1}, \mathbf{o}) = 1$, then the quality of the next state given source $\mathbf{o}$ is dependent on the next chosen option $\max_{\mathbf{o}' \in O} Q_{\mathbf{o}'}(\mathbf{s}_{i+1}, \mathbf{o}')$. Otherwise, the quality is given by the next state of the same source option $Q_{\mathbf{o}}(\mathbf{s}_{i+1}, \mathbf{o})$.

To update the termination network β, since the termination network controls the transition between supervisors, the goal is to maximize the quality $U(\mathbf{s}_1, \mathbf{o}_1)$ with regard to the first step in each indexing trials,

$$\mathbb{L}(\beta) = \sum_{\mathbf{s}_1 \in \mathbb{D}} U(\mathbf{s}_1, \mathbf{o}_1). \tag{11}$$

6 EXPERIMENT

In this section, we conduct extensive experiments to evaluate the proposed techniques by answering the following questions:



(a) TPC-H 1GB **(b) TPC-H 10GB** **(c) TPC-H 100GB** **(d) TPC-DS 1GB** **(e) TPC-DS 10GB** **(f) TPC-DS 100GB** **(g) JOB**

Figure 7: Relative workload cost of different index advisors

- **RQ1:** How does BALANCE reduce the workload execution cost, training overhead, and inference runtime, compared with other index advisors?
- **RQ2:** Can BALANCE generalize to dynamic workloads with different frequencies and query costs?
- **RQ3:** How does each component in BALANCE contribute to the overall performance? In particular, **RQ3.1:** Can the workload representation module improve BALANCE's performance? **RQ3.2:** How does the size of the pre-train dataset affect the workload representation module? **RQ3.3:** What is the impact of the transfer module in reducing workload cost and training overhead? **RQ3.4:** Does chunk segmentation affect BALANCE's superiority, especially in extreme cases (e.g., random workload shifts)?

6.1 Experiment Setup

Environment and Implementation. All the experiments are conducted with Python 3.7, PostgreSQL 12.5on 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.

Datasets. Since we focus on indexing for analytic workloads and do not consider index maintenance and update costs, we use the following three OLAP benchmarks: (1)TPC-H [29] is an open-source OLAP benchmark that contains eight tables and 61 columns; (2) TPC-DS [26] is an open-source OLAP benchmark that contains 25 tables and 429 columns; (3) JOB [20] is based on the IMDB dataset and contains 21 tables and 108 columns. For TPC-H and TPC-DS benchmarks, we generate three different data sizes (i.e., 1GB, 10GB, and 100GB) to study the effect of data volume. For JOB, we employ the default data volume of 7.5GB.

Workloads. To simulate dynamic workloads, we synthesize four workload chunks, i.e., $< W^1, \cdots, W^4 >$. In each chunk, we generate 300 workloads, each containing $|w|$ queries, by randomly selecting and populating query templates within the benchmark ($|w| = 14$ for TPC-H, $|w| = 40$ for TPC-DS and $|w| = 30$ for JOB) and assigning random frequencies (between 1 and 10000) to these queries. Each chunk adopts a different set of query templates, and a different distribution (i.e., with different random seeds) to draw query variants, values, and query frequencies. This is to reflect the *distant variety* of real workloads. Note that templates are randomly selected because we want to minimize constraints on the workload shift, e.g., we do not ask the workload stream to have a cyclic pattern, and we do not assume workload chunks to be a fixed set of templates, etc.

Unless otherwise stated (e.g., we will explore extremely random workload shifts in Section 6.7), workloads in each chunk are generated from a fixed set of query templates and query structures. Furthermore, we consider the following three settings of similar workloads.

`Setting 1: Query Variation.` As shown in Figure 6a, queries in different workloads are instances of the same templates with varying predicate values (i.e., TPC-H and TPC-DS) or the same predefined query structures with different selections (i.e., JOB). In many domains, query variation workloads are encountered when users interact with databases and issue queries using different attributes or values to retrieve specific information, tailoring the results to their needs. For example, customers of an e-commerce website may search for different product keywords or filter for different price ranges to compare products.

`Setting 2: Frequency Variation.` As shown in Figure 6b, the different workloads contain identical queries with varying query frequencies. Workloads of frequency variation occur when a fixed set of queries can express the user demands, and the query frequency is influenced by factors such as celebrities, seasonal trends, or specific events. For example, in social media, the user's search frequency of a particular celebrity tends to align with the fluctuations in social hot spots.

`Setting 3: A Combination of Query Variation and Frequency Variation.` As shown in Figure 6c, combining the above two settings, different workloads contain different instances of the same set of predefined query templates and query structures. Moreover, the frequencies of each query can vary. This setting is also typical in real scenarios. For example, on a stock trading application, users can search according to different stock codes, names, etc., and the query frequency fluctuates with market conditions.

Models. We compare BALANCE with five state-of-the-art competitors, including (1) DQN [18], which adopts the Deep Q-Network algorithm and requires indexing trials for each workload; (2) DRLindex [30, 31], which also adopts the Deep Q-Network algorithm and directly recommends indexes by training a network; (3) SWIRL [17], which employs proximal policy optimization algorithm and recommends indexes for workloads containing unseen queries; (4) Extend [32], which employs a recursive strategy and produces near-optimal workload cost for small index selection problems; (5) DB2Advisor [36], which maintains a sorted order of indexes and adds indexes to the final index configuration. As per [17], Extend [32] performs the best workload cost reduction, and SWIRL [17] is comparable with Extend in workload cost but with smaller inference runtime. We leverage the open-source implementation of the heuristic-based index advisors (i.e., DB2Advisor and Extend) [5], DQN [18], SWIRL [17] and we implement DRLindex as per the original paper [30, 31].

Training and Testing. Each experiment is repeated for three runs with different random seeds to make our analysis more reliable. In each run, we randomly draw 10 workloads from the fourth workload chunk W^4 as testing workloads and let all models make index recommendations. Note that DRLindex and SWIRL can make direct recommendations, while DQN needs to perform indexing trials on each testing workload. We allow DQN to iterate for 800 trial epochs, which we observe is sufficient for DQN to converge. Unless otherwise stated, in training BALANCE, from each chunk, we randomly draw 200 workloads (which are not testing workloads) and train four index advisor sub-modules with 2000 training epochs, as depicted in Section 3.2. To train DRLindex and SWIRL, we randomly draw 200 non-testing workloads from the fourth workload chunk and train them with 3000 training epochs. Using too many training workloads for these methods will dramatically increase the training complexity and hurt their performance. Hence, we only use the target workload chunk as training workloads. Extend, DB2Advisor, and DQN do not need training.

Other implementation details. The storage budget is randomly picked from a fixed range (500-10000MB for TPC-H 1GB and TPC-DS 1GB, 5000-100000MB for TPC-H 10GB and TPC-DS 10GB, 50000-1000000MB for TPC-H 100GB and TPC-DS 100GB, 500-12500MB for JOB). All benchmark-defined secondary indexes are removed for the following experiments. We adopt HypoPG [14] for the what-if optimizer. We use $|\mathcal{N}| = 300$ random workloads to pre-train the workload representation module and define the threshold τ^+ as the 20% quantile in the training set (details in Section 4).

Evaluation Metrics. We evaluate the index advisor's performance from three angles. (1) `Relative Workload Cost:` the ratio of workload execution cost with index to that without index. A smaller relative workload cost suggests a better indexing performance. (2) `Training Overhead:` for RL-base index advisors that need to be trained, training overhead is the time consumed from the start of training to model loss convergence. (3) `Inference Runtime:` the time consumption of the algorithms in choosing index configurations for each workload.

6.2 Performance Comparison

Relative workload cost. To study the performance of BALANCE (**RQ1**), we first report the relative workload cost produced by each method on the five datasets under the three settings in Figure 7. We have the following observations. (1) Compared with other learning-based competitors, BALANCE significantly reduces workload cost on each dataset under all three settings. On average, BALANCE reduces the workload cost by 15.27%, 38.62% and 10.03% on all datasets than DQN, DRLindex, and SWIRL, respectively. On the contrary, the competitors perform poorly in some situations. For example, DRLindex and DQN's performance is unsatisfying on TPC-DS dataset, e.g., the execution cost is still 90% of the cost without indexes, which means that the recommended indexes are not properly working. (2) Notably, BALANCE is capable of encoding

[5]https://github.com/hyrise/rl_index_selection

Table 2: Inference runtime comparison (seconds)

Dataset	DQN	DRLindex	SWIRL	Extend	DB2Advisor	BALANCE
TPC-H 1GB	6.25	30.18	0.47	77.70	4.04	**0.45**
TPC-H 10GB	8.53	38.84	0.55	89.43	4.69	**0.50**
TPC-H 100GB	12.69	49.80	0.72	103.41	5.34	**0.68**
TPC-DS 1GB	19.77	189.67	5.40	2369.23	14.02	**5.02**
TPC-DS 10GB	23.67	249.80	6.67	2782.33	20.91	**6.41**
TPC-DS 100GB	28.10	335.29	8.03	3290.45	27.57	**7.80**

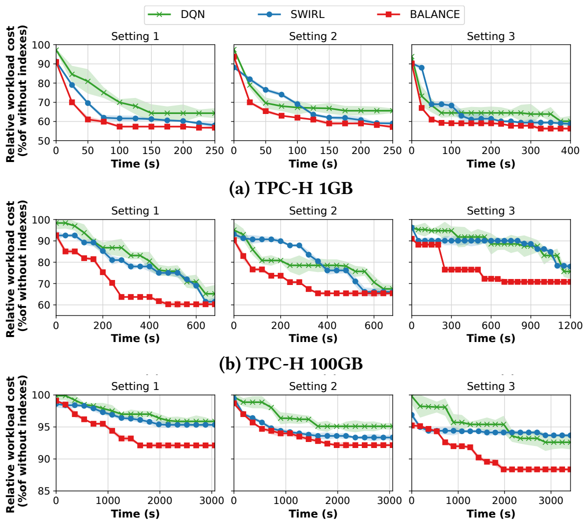
Table 3: Range of high, medium, low cost and frequency

	low	medium	high
Query Cost $cost(q, \emptyset)$	1-100000	100001-200000	200001-800000
Frequency f	1-3000	4000-6000	7000-10000

any query, from simple queries to queries with richer syntax. For example, template-18 queries in TPC-H benchmark include "index nested loop join" operator, which BALANCE effectively supports with strong performance. (3) BALANCE is comparable to Extend, which is considered near-optimal. On JOB dataset, BALANCE outperforms Extend by 26.47% on average. We want to emphasize that BALANCE is much faster than Extend in making the index recommendation, which will be shown below.

Inference runtime. Table 2 presents the average sum of inference runtime (10 test workloads are executed 3 times in each of the 3 settings) using different index advisors. We can see that BALANCE takes minimal inference runtime to make index recommendations. Firstly, the inference runtime of BALANCE is 2 to 3 orders of magnitude faster than Extend. This is because BALANCE directly recommends indexes by feeding the workload features to the network and performing forward inference in less than a second, while heuristic-based index advisors such as Extend require significant time to iteratively search for a satisfactory solution. Secondly, BALANCE is much faster than learning-based index advisors such as DQN and DRLindex. BALANCE reduces search time through action masking. On the other hand, methods like DQN and DRLindex have a time-consuming search process for index recommendations, especially when dealing with large databases. Thirdly, SWIRL is the second fastest index advisor because it also makes direct recommendations. Nonetheless, BALANCE is faster than SWIRL, which shows the efficiency of using a pre-trained workload representation module.

Training overhead. For DQN, SWIRL, and BALANCE with the least inference runtime, we further investigate their training overhead. In each run, we record the average relative workload cost of ten test workloads the above methods can achieve in each training epoch. Since the time for completing a training epoch varies among methods, for a fair comparison, we plot the workload cost of the ten testing workloads w.r.t the training time for three runs. As shown in Figure 8, we have three observations: (1) BALANCE achieves optimal workload cost while requiring a shorter training time, i.e., converges faster than other RL-based competitors. This indicates that BALANCE effectively transfers knowledge from the source policy via the transfer module, supporting faster training and reducing the overall training runtime. (2) BALANCE generally exhibits smaller variance, implying higher stability in predicting accurate indexes over different runs, which generalize better across

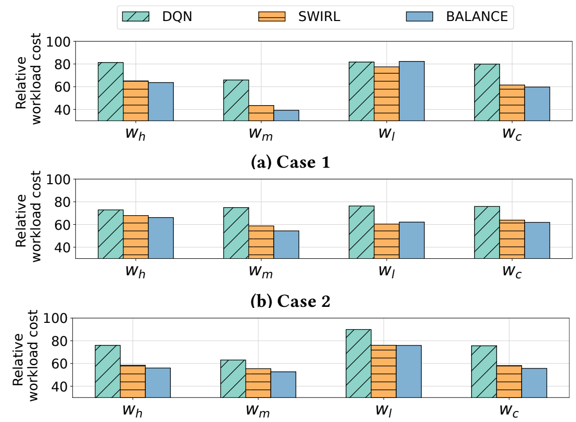


(a) TPC-H 1GB

(b) TPC-H 100GB

(c) TPC-DS 100GB

Figure 8: Average workload cost w.r.t the training time



(a) Case 1

(b) Case 2

(c) Case 3

Figure 9: Workload cost of BALANCE with different test cases

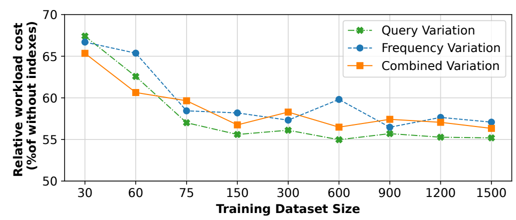


Figure 10: Average workload cost obtained by pre-training with different training data size on TPC-H 1GB dataset

different workloads. (3) When the database size or the number of tables increases, the training overhead of all methods increases. This is reasonable because a larger database or more tables result in a substantial increase in the time required for calling the what-if optimizer. But BALANCE experiences a comparatively smaller increase than DQN and SWIRL. For example, from TPC-H 1GB to TPC-H 100GB, to obtain 95% relative workload cost reduction, the training overhead increases from 100 sec to 700 sec for SWIRL, and 50 sec to 400 sec for BALANCE. Compared with DQN and SWIRL, BALANCE can achieve good performance with fewer trials through the transfer module, minimizing the usage of the what-if optimizer. Consequently, BALANCE demonstrates acceptable training overhead even on large and complex databases.

In summary, the comparative experiments show that BALANCE can achieve near-optimal index recommendation with the least inference runtime, and BALANCE significantly increases the training efficiency compared with other learning-based index advisors.

6.3 Generalization

The study the generalization of BALANCE (**RQ2**), we evaluate BALANCE using different testing workloads on TPC-DS 1GB. As listed in Table 3, we randomly generate queries and partition them into three categories, representing queries with high, medium, and low execution costs, respectively, denoted as Q_h, Q_m, and Q_l. We construct workloads w_h, w_m, w_l to contain only queries with high, medium, and low execution costs. And we use w_c to denote the union of all workloads. Formally, $w_c = \{w_h, w_m, w_l\}$, and $\forall(q_i, f_i) \in w_h, q_i \in Q_h$; $\forall(q_i, f_i) \in w_m, q_i \in Q_m$; and $\forall(q_i, f_i) \in w_l, q_i \in Q_l$.

Next, we consider three different cases to assign query frequencies in w_h, w_m, w_l. (1) `Case 1`: Assign high, medium, and low frequencies to queries with high, medium, and low execution cost categories, respectively. (2) `Case 2`: Assign low, medium, and high frequencies to queries with high, medium, and low execution cost categories, respectively. (3) `Case 3`: Assign medium, high, and low frequencies to queries with high, medium, and low execution cost categories, respectively.

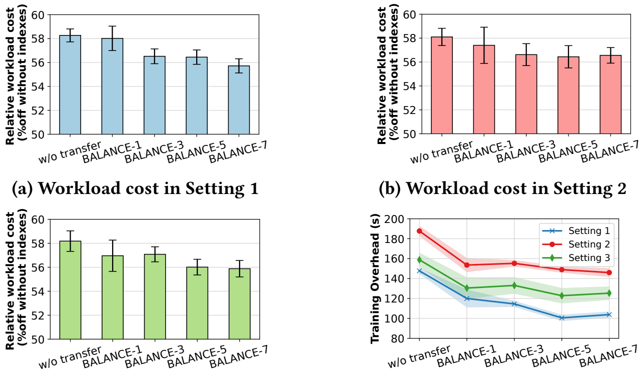
Figure 9 reports BALANCE's relative workload cost of w_c, w_h, w_m, w_l, compared with DQN and SWIRL. (1) Overall, BALANCE outperforms both SWIRL and DQN in w_c, w_h, w_m. This indicates that BALANCE is efficient for queries with high potential to be optimized because BALANCE can encode indexing aware characteristics of queries. (2) All methods perform better in Case 2 than in Case 3. This enhancement occurs because in Case 2, we assign a higher frequency to Q_l, leading to an increase in the optimization benefit of w_l.

6.4 Ablation on workload representation

To investigate the role of self-supervised workload representation learning (**RQ3.1**), we consider four variants of the workload representation while fixing other components in BALANCE. (1) `column`: The workload representation is a matrix denoted by the occurrence of the indexable attribute(s) in each query [30, 31]. (2) `BOO`: The workload representation is obtained by the Bag of Operator method [17]. (3) `codeBERT`: The workload representation is derived by feeding query text into a pre-trained codeBERT model [10]. CodeBERT is a well-trained program representation learned from conventional pre-training tasks. (4) `w/o CL (contrastive learning)`:

Table 4: Related workload cost by variants of workload representation on TPC-H 1GB dataset.

Setting	column	BOO	codeBERT	w/o CL	BALANCE
Setting 1	59.69	59.92	58.92	57.72	**56.53**
Setting 2	60.64	58.81	58.45	57.55	**56.64**
Setting 3	60.53	59.73	58.32	58.21	**57.42**



(a) Workload cost in Setting 1 **(b) Workload cost in Setting 2**

(c) Workload cost in Setting 3 **(d) Training overhead**

Figure 11: Workload cost and training overhead by variants of transfer module on TPC-H 1GB dataset

The workload representation defined by Equation 3, including the operator encoding and the value encoding.

We compare BALANCE with the above variants. The average workload cost is shown in Table 4. We have the following observations. (1) BALANCE achieves a workload cost reduction of 1.67% compared to w/o CL. This observation reveals the importance of incorporating indexing performance-related information into workload representations via unsupervised contrastive learning. (2) Parsing workload representation from the query plan (i.e., BOO, w/o CL, and BALANCE) outperforms other variants (i.e., column and codeBERT), indicating that more effective information can be extracted from the query plan. Simply relying on the information in the query text is unsuitable for the index selection problem, as the query plan provides additional details about query execution and is updated during the index selection process. (3) Workload representation by column is comparable to workload representation by codeBERT. While masked language pre-training excels in NLP, this observation suggests that treating queries as text with masked language pre-training is ineffective for representing ISP workloads.

6.5 Impact of Pre-training Data Size

To uncover the impact of training data size in pre-training the workload representation module (**RQ3.2**), we report the relative workload cost, averaged over three runs, with respect to different sizes of the pre-training data $\mathcal{N}$ on the TPC-H 1GB dataset. We have the following observations from Figure 10. (1) The workload cost decreases as the pre-train dataset size increases. This is reasonable because large training data provide more diverse workload samples, and the workload representation module can better discriminate workloads with different indexing properties in contrastive learning. The workload representation will be more fine-grained and benefit the ISP. (2) BALANCE's performance plateaus when $|\mathcal{N}| \geq 150$, indicating that the performance of BALANCE does not require a very large pre-training dataset size. A moderate size (e.g., $|\mathcal{N}| = 300$) can ensure good indexing performance with fewer resources. Considering the trade-off between performance and training cost, choosing a moderate size for the pre-training dataset is recommended.

6.6 Ablation on the Transfer Module

To verify the importance of transfer learning in BALANCE (**RQ3.3**), we compare with the following variants. (1) `w/o transfer`: We remove the transfer module and train BALANCE using workloads from the target workload chunk. (2) `BALANCE-X`: We construct $X+1$ workload chunks, where $X = 1, 3, 5, 7$, and the former X index advisor sub-modules are considered source models to supervise the last index advisor for workload chunk W^{X+1}.

Figure 11 reports the workload cost and the training overhead of variants under different settings on TPC-H 1GB. The experimental results reveal the following: (1) Incorporating the policy transfer mechanism reduces the training overhead and relative workload cost. This improvement is due to the enhanced sample efficiency of reinforcement learning through the guidance of source models, resulting in more accurate index recommendations by the target model. (2) As the number of source models increases, both the relative workload cost and training overhead show a downward trend. (3) Compared with without transfer learning, BALANCE-1 exhibits a larger variance across the three runs. This outcome can be attributed to the fact that when there is only one source model, the target model is more heavily influenced by this single model, leading to an unstable performance. (4) Under all three settings, employing 5 source models produces satisfactory performance, and employing more source models has a negligible improvement. This observation highlights BALANCE's *long-term adaptability*, i.e., BALANCE only needs to maintain a small number of source models (i.e., resource requirements and computational demands are relatively constant) to achieve good transfer effects for workloads that continue in a longer time period.

6.7 Impact of Chunk Segmentation

To extensively investigate BALANCE's capability in more extreme scenarios, e.g., workloads are completely random over time, or no pattern in chunks can be exploited (**RQ3.4**), we synthesize a different flow of workloads on TPC-DS 1GB dataset. In the first timestamp, we randomly select 40 templates. In the following timestamps, we substitute a random portion of query templates in the previous workload. The predicate values and frequencies of the queries are randomized. We repeat the above for 30 timestamps.

To segment the workload flow into chunks, we first adopt a `Split-X%` approach, which merges subsequent timestamps into a chunk if the difference of query templates is less than $X\%$. For example, suppose the query templates in current chunk t are denoted as a set $\mathcal{P}_t$, and the query templates in the next timestamp m are denoted as $\mathcal{P}_m$. If $(\mathcal{P}_t \cap \mathcal{P}_m)/|\mathcal{P}_m| \geq (1 - X\%)$, where $X\%$ is the given difference threshold, then timestamp m is merged into chunk t. Otherwise, a new chunk $t+1$ that contains m is constructed.

Table 5: Workload cost under different splitting methods

Method	Split-10%	Split-20%	Split-40%	Split-60%	w/o pattern
DQN	85.31	77.47	72.36	79.08	78.10
Extend	64.34	**64.39**	**62.09**	**56.56**	**62.25**
DB2Advisor	68.46	68.78	63.76	58.90	66.69
SWIRL	66.10	68.37	65.87	63.28	70.91
BALANCE	**64.03**	67.55	64.39	59.53	67.76

Table 6: Workload cost w.r.t. storage budgets in Setting 3

Budget	DQN	DRLindex	SWIRL	DB2Advisor	Extend	BALANCE
500 MB	71.59	76.33	67.94	68.48	65.75	**65.21**
1000 MB	69.25	70.64	62.90	65.53	**61.00**	61.80
5000 MB	64.55	66.51	55.83	58.99	**54.12**	54.87
10000 MB	58.62	61.85	54.74	57.12	**52.29**	52.72

Clearly, chunks segmented on the workload flow in this subsection can accommodate larger workload changes than the three settings in previous subsections. A larger X indicates a larger variety in each chunk. We then adopt a `w/o pattern` approach, by simply putting 300 workloads into a chunk. Then, workloads in each chunk undergo severe changes, and no pattern can be exploited. We want to point out that all approaches in this paper do not need future workload forecasting to segment workload chunks, and the split-X% percentage is given by the user, which relies on the application scenario, i.e., workload shift and computational resource.

Table 5 reports the workload cost, and we have the following observations. (1) Performance of BALANCE generally decreases as X increases, while DQN, DB2Advisor and Extend are relatively stable. This is attributed to the large inconsistency between BALANCE's training set (i.e., a small set of samples drawn from the chunk) and the testing set (i.e., the other workloads in the same chunk). (2) Nevertheless, BALANCE outperforms DQN and SWIRL on different X. BALANCE is even comparable to DBA2Advis in Split-60%, which is a surprisingly good performance, considering BALANCE's ability to make direct recommendations with little inference runtime. (3) BALANCE's performance degradation is acceptable (i.e., comparable to DB2Advisor) on w/o pattern chunks. It shows that, despite the workloads being entirely distinct, common knowledge can still be transferred and benefit index selection strategies.

6.8 Impact of Storage Budget

To evaluate the impact of the storage budgets, we conducted experiments on the TPC-H 1 GB dataset under different storage budgets (e.g., 500MB, 1000MB, 5000MB, 10000MB).

Table 6 reports the relative workload cost under different storage budgets in Setting 3. The experimental results reveal the following conclusions. (1) As the storage budget increases, BALANCE more significantly decreases the workload cost, as a larger budget allows BALANCE to include a greater number of indexes in the index configuration. (2) BALANCE outperforms other learning-based competitors across diverse storage budgets. BALANCE considers the impact on index storage in the RL framework and enables the model to identify indexes with high efficiency and low storage consumption. In contrast, DQN and DRLindex prioritize high benefits, reaching storage budgets after selecting several indexes and limiting further choices. (3) BALANCE demonstrates comparable performance to Extend and even outperforms it under a 500MB storage budget, indicating that BALANCE effectively adapts to various storage budgets, achieving near-optimal performance with short inference time. This adaptability highlights BALANCE's potential across various resource constraints and dynamic scenarios.

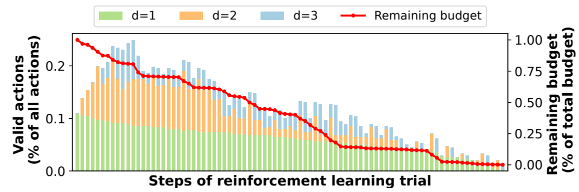


Figure 12: Valid action w.r.t step of RL trial

6.9 Effectiveness of Action Masking

To investigate the effectiveness of action masking , we track the percentage of valid actions w.r.t. index widths $d = 1$, 2, or 3 at each step during a test workload RL trial on TPC-H 1GB dataset in Figure 12. We have the following observations. (1) Initially, approximately 0.10% of the actions are valid, gradually increasing to 0.25% as the trial progresses. This is attributed to many multi-attribute indexes (i.e., $d = 2, 3$) being deleted due to invalid preconditions, and as the trial progresses, more single-attribute indexes (i.e., $d = 1$) are added to the index configuration, previously excluded multi-attribute indexes become valid actions. (2) As the trial continues and the remaining storage budget decreases, more indexes become invalid, mainly because they exceed the permissible size limit. (3) Through the entire trial process, most valid actions consist of indexes with widths of 1 and 2, and the fraction of valid actions did not surpass 0.25%. This outcome highlights the efficacy of the action masking in effectively constraining the action space.

7 CONCLUSION AND FUTURE WORK

We propose BALANCE for dynamic and heterogeneous workloads in real-world scenarios with a novel transfer RL-based framework and an unsupervised workload representation learning method. Experimental results show that our method outperforms existing approaches in efficiency and effectiveness. Several open problems lie ahead in the domain of index advisors. The first is to enhance the index advisor's generalization across various database systems, considering differences in schema, query optimization strategies, and data distributions. The second is to extend the index advisor to cater to cloud-based or distributed databases, focusing on elastic resource provisioning and optimized communication in the distributed architecture.

ACKNOWLEDGEMENTS

Chen Lin is the corresponding author. Chen Lin is supported by the Natural Science Foundation of China (No.62372390) and CCF-Huawei Populus Grove Fund. Guoliang Li is supported by the National Key R&D Program of China (2023YFB4503600), NSF of China (61925205, 62232009, 62102215), and Zhongguancun Lab. Zhifeng Bao is supported in part by ARC DP240101211.

REFERENCES

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38.

[2] André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, David Silver, and Hado van Hasselt. 2017. Successor Features for Transfer in Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 4055–4065.

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

[4] Nicolas Bruno and Surajit Chaudhuri. 2007. An Online Approach to Physical Design Tuning. In *2007 IEEE 23rd International Conference on Data Engineering*. 826–835.

[5] Tim Brys, Anna Harutyunyan, Matthew E. Taylor, and Ann Nowé. 2015. Policy Transfer using Reward Shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS '15)*. 181–188.

[6] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceedings of the 23rd International Conference on Very Large Data Bases*. 146–155.

[7] Surajit Chaudhuri and Vivek R. Narasayya. 1998. AutoAdmin 'What-if' Index Analysis Utility. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. 367–378.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning*, Vol. 119. 1597–1607.

[9] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science* 41, 6 (1990), 391–407.

[10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 1536–1547.

[11] Fernando Fernández and Manuela M. Veloso. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06)*. 720–727.

[12] Michael Hammer and Arvola Chan. 1976. Index Selection in a Self-Adaptive Data Base Management System. In *Proceedings of the 1976 ACM SIGMOD International Conference on Management of Data*. 1–8.

[13] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 9726–9735.

[14] HypoPG. 2015. https://github.com/HypoPG/hypopg.

[15] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. 2020. A Survey on Contrastive Self-supervised Learning. *arXiv Preprint* (2020). https://arxiv.org/abs/2011.00362

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

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

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

[19] Romain Laroche and Merwan Barlier. 2017. Transfer Reinforcement Learning with Shared Dynamics. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 2147–2153.

[20] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215.

[21] Siyuan Li and Chongjie Zhang. 2018. An Optimal Online Method of Selecting Source Policies for Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 3562–3570.

[22] 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.

[23] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 International Conference on Management of Data*. 631–645.

[24] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv Preprint* (2013). https://arxiv.org/abs/1312.5602

[26] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The Making of TPC-DS. In *Proceedings of the 32nd International Conference on Very Large Data Bases*. 1049–1058.

[27] 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 *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 600–611.

[28] Gregory Piatetsky-Shapiro. 1983. The Optimal Selection of Secondary Indices is NP-Complete. *SIGMOD Rec.* 13, 2 (1983), 72–75.

[29] Meikel Pöss and Chris Floyd. 2000. New TPC Benchmarks for Decision Support and Web Commerce. *SIGMOD Rec.* 29, 4 (2000), 64–71.

[30] Zahra Sadri, Le Gruenwald, and Eleazar Leal. 2020. DRLindex: deep reinforcement learning index advisor for a cluster database. In *Proceedings of the 24th Symposium on International Database Engineering & Applications*. 11:1–11:8.

[31] Zahra Sadri, Le Gruenwald, and Eleazar Leal. 2020. Online Index Selection Using Deep Reinforcement Learning for a Cluster Database. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*. 158–161.

[32] Rainer Schlosser, Jan Kossmann, and Martin Boissier. 2019. Efficient Scalable Multi-attribute Index Selection Using Recursive Strategies. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 1238–1249.

[33] 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

[34] Vishal Sharma and Curtis E. Dyreson. 2022. Indexer++: workload-aware online index tuning with transformers and reinforcement learning. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. 372–380.

[35] Richard S. Sutton and Andrew G. Barto. 1998. Reinforcement Learning: An Introduction. *IEEE Trans. Neural Networks* 9, 5 (1998), 1054–1054.

[36] 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 16th International Conference on Data Engineering*. 101–110.

[37] Kyu-Young Whang. 1987. Index Selection in Relational Databases. In *Foundations of Data Organization*. 487–500.

[38] 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 2022 International Conference on Management of Data*. 1528–1541.

[39] Tianpei Yang, Jianye Hao, Zhaopeng Meng, Zongzhang Zhang, Yujing Hu, Yingfeng Chen, Changjie Fan, Weixun Wang, Wulong Liu, Zhaodong Wang, and Jiajie Peng. 2020. Efficient Deep Reinforcement Learning via Adaptive Policy Transfer. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. 3094–3100.

[40] Zonghan Yang, Yong Cheng, Yang Liu, and Maosong Sun. 2019. Reducing Word Omission Errors in Neural Machine Translation: A Contrastive Learning Approach. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Vol. 1. 6191–6196.

[41] 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 *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 2196–2208.

[42] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. 2020. Transfer Learning in Deep Reinforcement Learning: A Survey. *arXiv Preprint* (2020). https://arxiv.org/abs/2009.07888