

# WISK: A Workload-aware Learned Index for Spatial Keyword Queries

YUFAN SHENG, University of New South Wales, Australia

XIN CAO\*, University of New South Wales, Australia

YIXIANG FANG, The Chinese University of Hong Kong, Shenzhen, China

KAIQI ZHAO, The University of Auckland, New Zealand

JIANZHONG QI, The University of Melbourne, Australia

GAO CONG, Nanyang Technological University, Singapore

WENJIE ZHANG, University of New South Wales, Australia

Spatial objects often come with textual information, such as Points of Interest (POIs) with their descriptions, which are referred to as geo-textual data. To retrieve such data, spatial keyword queries that take into account both spatial proximity and textual relevance have been extensively studied. Existing indexes designed for spatial keyword queries are mostly built based on the geo-textual data without considering the distribution of queries already received. However, previous studies have shown that utilizing the known query distribution can improve the index structure for future query processing. In this paper, we propose WISK, a learned index for spatial keyword queries, which self-adapts for optimizing querying costs given a query workload. One key challenge is how to utilize both structured spatial attributes and unstructured textual information during learning the index. We first divide the data objects into partitions, aiming to minimize the processing costs of the given query workload. We prove the NP-hardness of the partitioning problem and propose a machine learning model to find the optimal partitions. Then, to achieve more pruning power, we build a hierarchical structure based on the generated partitions in a bottom-up manner with a reinforcement learning-based approach. We conduct extensive experiments on real-world datasets and query workloads with various distributions, and the results show that WISK outperforms all competitors, achieving up to 8× speedup in querying time with comparable storage overhead.

CCS Concepts: • **Information systems** → **Location based services**; **Query optimization**.

Additional Key Words and Phrases: learned index, geo-textual data, spatial keyword query

## ACM Reference Format:

Yufan Sheng, Xin Cao, Yixiang Fang, Kaiqi Zhao, Jianzhong Qi, Gao Cong, and Wenjie Zhang. 2023. WISK: A Workload-aware Learned Index for Spatial Keyword Queries. *Proc. ACM Manag. Data* 1, 2, Article 187 (June 2023), 27 pages. <https://doi.org/10.1145/3589332>

\*Xin Cao is the corresponding author.

Authors' addresses: Yufan Sheng, University of New South Wales, Sydney, Australia, [yufan.sheng@unsw.edu.au](mailto:yufan.sheng@unsw.edu.au); Xin Cao, University of New South Wales, Sydney, Australia, [xin.cao@unsw.edu.au](mailto:xin.cao@unsw.edu.au); Yixiang Fang, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China, [fangyixiang@cuhk.edu.cn](mailto:fangyixiang@cuhk.edu.cn); Kaiqi Zhao, The University of Auckland, Auckland, New Zealand, [kaiqi.zhao@auckland.ac.nz](mailto:kaiqi.zhao@auckland.ac.nz); Jianzhong Qi, The University of Melbourne, Melbourne, Australia, [jianzhong.qi@unimelb.edu.au](mailto:jianzhong.qi@unimelb.edu.au); Gao Cong, Nanyang Technological University, Singapore, Singapore, [gaocong@ntu.edu.sg](mailto:gaocong@ntu.edu.sg); Wenjie Zhang, University of New South Wales, Sydney, Australia, [wenjie.zhang@unsw.edu.au](mailto:wenjie.zhang@unsw.edu.au).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

2836-6573/2023/6-ART187 \$15.00

<https://doi.org/10.1145/3589332>

Keywords:  $q_1.kws = \{Park, BBQ\}$  and  $q_2.kws = \{Hotel, Pizza\}$

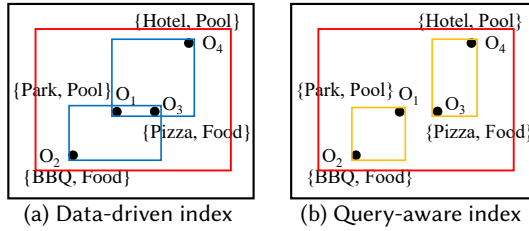


Fig. 1. Data-driven indexes vs query-aware indexes

## 1 INTRODUCTION

The worldwide mobile internet use has already surpassed desktop use since late 2016<sup>1</sup>. In such a mobile internet, massive **geo-textual data** with spatial and textual attributes are generated, e.g., Points of Interest (POIs) in Google Maps are associated with geo-locations and descriptive texts. Managing and retrieving geo-textual data at scale has attracted much attention. In recent years, the **spatial keyword queries** [7, 12, 13, 15, 16, 20, 26, 58, 74] have been extensively studied, which take a location and a set of keywords as arguments and return objects based on different definitions of spatial proximity and textual relevance. **Spatial keyword indexes** are developed to process such queries efficiently by incorporating the techniques of indexing spatial objects and documents.

However, state-of-the-art spatial keyword indexes still have several drawbacks. First, as shown in previous studies [10, 12, 41], no existing index can work efficiently for all data distributions, and there is no single approach that dominates all others. Second, traditional indexes may have some parameters to be set with fixed values across the entire input data space. For example, CDIR-tree [15] needs a parameter to balance the importance of spatial proximity and textual relevancy. Since the query and data distributions vary in different areas, it is hard to select a single parameter value that fits all distributions. Third, no indexes have considered utilizing the query workload. Previous works [6, 55] have shown that certain data regions could be much more heavily queried. For spatial keyword queries, both the query keyword distribution and query location distribution can be various over different spatial regions. Thus, utilizing the known query distribution could further optimize the index structure for future query processing [18, 45]. For example, as shown in Figure 1, a spatial keyword query workload includes  $q_1$  and  $q_2$  with the same query region (red rectangle) and different keywords. Existing data-driven indexes (Figure 1a) store objects with close spatial distances and large textual similarities into a partition (enclosed by blue rectangles), and both queries have to check two partitions containing four objects. If the four objects are grouped in a different way to adapt for the query keywords, both queries only need to check two objects in a single partition. Query-aware index can be expected to achieve better performance on future queries following similar known query distributions.

Motivated by these observations, in this work, we propose a novel Workload-aware learned Index for Spatial Keyword queries (WISK). The objective is to learn an index structure using both the spatial and textual information such that the processing cost for the known query workload using this index is minimized. We focus on the spatial keyword range query workload.

There exist some learned indexes for spatial query processing, which can also be classified into *data-driven* indexes (such as ZM [62], LISA [38], and RSMI [51]) and *query-aware* indexes (such as Flood [45] and Tsunami [18]). These learned spatial index structures are not suitable for processing spatial keyword queries directly, because they only use spatial attributes for index learning. Concurrent with our work, a learned index has been proposed for spatial keyword

<sup>1</sup>Mobile vs. Desktop Internet Usage: <https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics/>

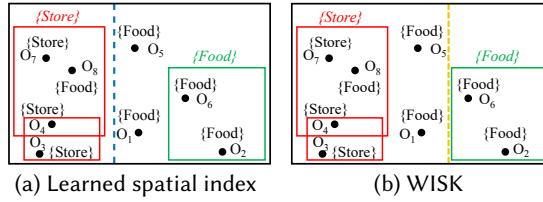


Fig. 2. Learning without and with textual information

querying [19]. It uses spatial attributes for index learning first and then creates the textual index, and thus it cannot well learn the spatial and textual correlations for building the index.

The key challenge of learning a spatial keyword index is how to capture the data and query distribution considering both the structured spatial information and unstructured textual attributes during index learning, and make use of the distribution captured to partition the data objects such that more irrelevant partitions can be filtered out during query processing, thus improving the querying efficiency. One simple method is to learn a spatial index first, which does not consider keywords, and then build an inverted file to manage the textual information within each partition of the index. Our experiments show that this has poor performance. The reason could be briefly explained in Figure 2, with a dataset containing 8 objects and a workload with 3 queries. The learned spatial indexes put fewer objects into the left partition which contains more queries to reduce costs. However, such partitions are worse than that in Figure 2b, when objects without query keywords can be ignored during query processing using inverted files. Figure 2a has 3 query-relevant objects (Store) in the left partition and 4 query-relevant objects (Food) in the right partition, and thus the number of checks required is  $2 \times 3 + 1 \times 4 = 10$ . But Figure 2b only needs  $2 \times 3 + 1 \times 2 = 8$  checks.

It can be observed that the query cost largely depends on how the objects are partitioned. We first formulate a problem of finding  $k$  partitions with the optimal cost for a given query workload. We show the NP-hardness of this problem by a reduction from the MaxSkip partitioning problem [56, 73]. To learn good partitions, we design a cost estimation method considering both spatial and textual information of the query and data, based on trained models that can approximate the Cumulative Distribution Function (CDF) of geo-textual data. Then, we propose a heuristic algorithm and use Stochastic Gradient Descent (SGD) [53] to generate the  $k$  partitions.

If there is only one-level of partitions, we need to check many partitions irrelevant to the query. Hence, we further group the partitions into a hierarchy as do most indexes. A simple way is to adopt the method in CDIR-tree [15] for building the tree in a bottom-up manner. However, it is hard to select the weights of the spatial proximity and textual relevance between partitions. We show that it might even have worse query time in experiments. Instead, we propose to pack the nodes level by level, and model the one-level node packing problem as a sequential decision-making process. In particular, we develop a reinforcement learning [29] algorithm to find the optimal packing for each level and build the index in a bottom-up manner by considering the query workload.

In summary, we make the following main contributions:

- We propose a query-aware learned index named WISK considering spatial and textual attributes simultaneously.
- To generate the leaf nodes of WISK, we define an optimal partitioning problem and show its NP-hardness. We propose a heuristic algorithm to solve the problem using machine learning techniques.
- To build the hierarchy of WISK, we propose to pack the nodes level by level in a bottom-up manner, and we treat the node packing as a sequential decision-making process. We develop a solution based on reinforcement learning.

- We perform a comprehensive empirical study using real-world datasets and synthetic query workloads with various distributions. The results show that WISK outperforms the state-of-the-art spatial keyword indexes consistently in terms of efficiency, achieving up to  $8\times$  times speedup while having a comparable index size.

## 2 PRELIMINARIES

### 2.1 Problem Statement

We consider a geo-textual dataset  $D$  where each data object, i.e., a geo-textual object  $o \in D$ , has a point location denoted as  $o.loc$  and a text description denoted as  $o.kws$ , which is available from a range of source [14]. For ease of discussion, we assume two-dimensional coordinates in Euclidean space to represent  $o.loc$ , although our proposed techniques can generalize to multi-dimensional spaces easily. The text description  $o.kws$  is represented as a set of keywords, e.g., tags indicating the functionality of a POI. We aim to process *spatial keyword range queries* over  $D$ .

**DEFINITION 1 (SPATIAL KEYWORD RANGE (SKR) QUERY).** An SKR query  $q$  is represented by a pair  $(q.area, q.keys)$  where  $q.area$  and  $q.keys$  denote a spatial region and a set of keywords, respectively. The result of  $q$ ,  $q(D) = \{o \in D \mid o.loc \text{ in } q.area, o.kws \cap q.keys \neq \emptyset\}$ , is a subset of  $D$  that includes all objects within the query region containing at least one query keyword.

Here, we use a rectangular query region. Our techniques can be easily extended to handle other shapes (e.g., circles) by an extra filtering after querying with the bounding rectangle.

**Problem.** Our goal is to learn an index structure that can efficiently process SKR queries utilizing the distributions of the geo-textual data and the given query workload.

### 2.2 Reinforcement Learning

Reinforcement learning (RL) [29] is a machine learning technique where an agent learns from feedback obtained from trial-and-error interactions with an environment. It has been shown to be effective for sequential decision-making problems [54, 78].

RL formulation is based on the Markov Decision Process (MDP) [50]. An MDP has four components: a set of states  $S$ , a set of actions  $A$ , transition probabilities  $P$ , and rewards  $R$ . At some state  $s \in S$ , an agent may take an action  $a \in A$ . As a result, there is a probability  $P_a(s, s')$  that the agent transits to state  $s'$ , and a reward  $R_a(s, s')$  is received from such an action and state transition. The goal of the agent is to learn a policy function  $\pi : A \times S \rightarrow [0, 1]$ , i.e., the probability of taking action  $a$  at state  $s$ , such that the cumulative reward of state transitions is maximized. Figure 3 shows the basic workflow of RL. The environment connects to the agent via perception and action and it offers the agent the possible action choices based on the current state of the agent. The agent learns its policy based on rewards accumulated from interactions with the environment. Its learning process stops when a terminal state is reached.

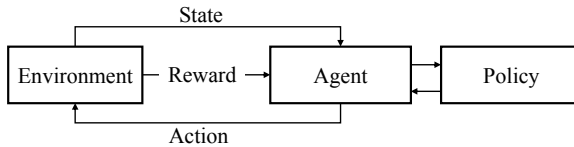


Fig. 3. The typical RL learning framework

Q-learning [67] is a commonly used value-based policy learning algorithm, which learns the value of an action given a state. It learns a policy that maximizes the value of a so-called Q-function,  $Q(s, a)$ , i.e., the overall expected reward when an agent plays following the policy [43]. State-of-the-art RL models such as Deep-Q-Network (DQN) [44] use a deep neural network  $Q(s, a; \theta)$  with

parameters  $\theta$  to estimate the value of the Q-function  $Q(s, a)$ . Once  $Q(s, a; \theta)$  is trained, it can be used for decision-making for future events.

### 3 INDEX OVERVIEW

WISK consists of two parts: (1) learn an optimal data layout for the given query workload, and (2) create an index based on that layout.

Query processing on an index that partitions objects into clusters typically involves two costly operations: filtering and verification. Irrelevant partitions are filtered out, and objects in the remaining partitions are verified. Thus, in WISK we first aim to learn an optimal partition of the geo-textual objects, such that for the given SKR query workload we can achieve the minimum query processing costs computed using both the filtering and verification costs. However, given a large dataset, the number of possible partitions of the objects is extremely huge, and it is hard to learn an optimal partition. We propose to simplify the problem: we divide the 2D space into disjoint partitions to obtain an optimal spatial layout. We will give the detail in Section 4.

If there is only one layer of the index, we need to check all partitions to see if they are relevant to the query, leading to a high filtering cost. We can organize the partitions obtained in the first step into a tree structure to build the final index such that the query processing cost can be further optimized. This is a type of combinatorial optimization task. We propose to pack the partitions level by level, and view the one-level packing problem as a sequential decision-making process, which can be solved by reinforcement learning. We will present the detail of this step in Section 5.

The framework of WISK is shown in Figure 4. A leaf node contains a number of objects, a minimum bounding rectangle (MBR) of the objects, and an inverted file to index the objects in this node. A non-leaf node contains pointers to its child nodes, an MBR for all child nodes, and a bitmap to index keywords appearing in its sub-tree. Here, the bitmap is used due to its much smaller size.

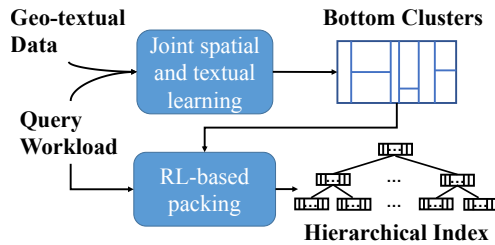


Fig. 4. The WISK framework

**Index construction.** Algorithm 1 summarizes the construction process of WISK, which consists of two main steps. Step 1 (lines 1 and 2) is to construct the bottom clusters. A high-quality partition of bottom clusters should result in a low query cost given a query workload. We train machine learning models to approximate the Cumulative Distribution Function (CDF) of geo-textual objects. Then, we define the cost estimation function based on the learned CDFs and make it differentiable, such that we can use stochastic gradient descent (SGD) to learn the optimal partitions. Step 2 (lines 3 and 4) is to construct the hierarchy of WISK by a bottom-up packing of the bottom clusters. Our goal is to minimize the filtering cost when an SKQ query is processed by this hierarchy. We model the packing process in one level as an MDP and apply RL to solve the problem. Based on the given query workload, we design a reward to measure the reduction of the filtering cost given a packing decision, and we train a model to predict the reward, which can be used to guide packing the bottom clusters level by level. The RL approach can also be used to group objects into bottom clusters. However, due to a large number of objects, this will require huge amounts of states during the RL procedure, and both the training time and the performance would be unacceptable [60].

---

**Algorithm 1: WISK Construction**


---

**Input:**  $Q$ , the workload;  $D$ , the dataset;  $C$ , the cost function

**Output:**  $I$ , a learned WISK index

```

1  $KM \leftarrow \text{MLModel}(D)$ ;                               /* Learn CDF models of objects */
2  $G \leftarrow \text{SGDPartition}(Q, C, KM)$ ;                 /* G minimizes C */
3  $RM \leftarrow \text{RLTrain}(G, Q)$ ;                       /* Learn a model RM based on G, Q */
4  $I \leftarrow \text{Packing}(RM, G)$ ;                       /* Group G by using RM */
5 return  $I$ ;

```

---

**Query processing.** Our query algorithm is similar to those using traditional spatial keyword indexes. Given a SKR query  $q$ , it traverses WISK in a breath-first manner starting from the root. A queue  $Q$  is used to manage the nodes visited. For every non-leaf node visited, only the child nodes whose MBRs overlap with  $q.area$  and contain some query keywords are added to  $Q$  for future verification. When a leaf node (i.e., the bottom cluster) is reached, we use its inverted file to fetch the query-relevant objects and return those in the query region.

#### 4 PARTITIONING OPTIMIZATION

A core problem in WISK construction is to form partitions (i.e., bottom clusters) for cost minimization over the query workload. In this section, we model the query cost, define an optimal partition problem, show the NP-hardness of the problem, and present a heuristic algorithm for the problem.

##### 4.1 Cost Model

We model the time cost  $C(q_i)$  to process an SKR query  $q_i$  over a set of bottom clusters  $G$  as a linear combination of (1) the cost to scan all bottom clusters to find a subset  $G_i \subset G$  that overlap with  $q_i.area$  and containing at least one keyword in  $q_i.kws$ , and (2) the cost to examine the inverted file in each cluster  $c \in G_i$  and find the objects in  $q_i.area$  and contain at least one query keyword. Eq. 1 formalizes the cost, where  $|G|$  denotes the total number of clusters, and  $\sum_{c \in G_i} |O_c|$  denotes the number of objects in  $G_i$  that contains at least one query keyword. In particular,  $w_1$  measures the time cost for checking (1) if the MBR of a cluster intersects with the query region, and (2) if the cluster contains some query keywords by scanning the textual index of the cluster. Both checks are independent of the cluster size. Meanwhile,  $w_2$  measures the time cost to perform the same checks but at the object level. Following recent studies [18, 77], we use fixed values for these parameters.

$$C(q_i) = w_1|G| + w_2 \sum_{c \in G_i} |O_c| \quad (1)$$

**Example 4.1:** Figure 5 illustrates this cost function. Suppose that the red and green points represent objects that contain keywords  $k_1$  and  $k_2$  respectively. There are two queries, and  $q_1.kws = \{k_1\}$  and  $q_2.kws = \{k_2\}$ . If all objects are in a cluster (i.e., no partitioning, Figure 5a), according to Eq. 1, the two queries incur a cost of  $2(w_1 + 4w_2) = 2w_1 + 8w_2$ . This is because there is only one cluster, and each query needs to check four objects containing the query keywords (i.e., four red points for  $k_1$  and four green points for  $k_2$ ). If the space is split forming two clusters of five and three points each (Figure 5b), the cost of  $q_2$  and  $q_1$  will become  $2w_1 + 2w_2$  (checking two clusters and two green

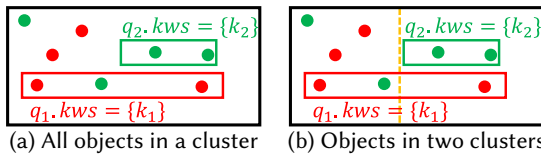


Fig. 5. Partitioning the space increases the number of clusters, which leads to a larger cluster scanning cost, but also potentially a lower object scanning cost.

points) and  $2w_1 + 4w_2$  (checking two clusters and four red points), which sum up to  $4w_1 + 6w_2$ . The partitioning may lead to an overall lower query cost if  $w_2$  dominates the cost.

## 4.2 The Optimal Partitioning Problem

We formulate an optimal partition problem to find a set of clusters that minimize the query cost over a given set of queries.

**PROBLEM 1 (OPTIMAL PARTITIONING).** *Given a dataset  $D = \{o_1, o_2, \dots, o_n\}$  and a query workload  $W = \{q_1, q_2, \dots, q_m\}$ , we aim to find an optimal partition, i.e., a set of  $k$  clusters  $G = \{c_1, c_2, \dots, c_k\}$  where (1) each object belongs to exactly one cluster, i.e.,  $\bigcup_{c_i \in G} c_i = D$ , and  $\forall_{c_i, c_j \in G} c_i \cap c_j = \emptyset$ , and (2) the total cost,  $\sum_{q_i \in W} C(q_i)$ , is minimized, where  $C(q_i)$  (Eq. 1) is the cost of  $q_i$ .*

**4.2.1 Problem Analysis.** We proceed to show that the optimal partitioning problem is NP-hard by reducing from the MaxSkip partitioning problem, which has been shown to be NP-hard [56, 73].

**THEOREM 4.1.** *Problem 1 is NP-hard.*

**PROOF.** We first briefly introduce the MaxSkip partitioning problem, which arises from big data analytics systems. Let  $Q$  be a collection of queries. Consider a set of partitions  $P = \{p_1, p_2, \dots, p_k\}$  where each partition is a collection of tuples, and the size of each partition is larger than a minimum size bound  $b$ . A big data analytics system can prune a partition  $p_i$  if none of the tuples in this partition satisfies a query  $q \in Q$  when processing  $q$ . A cost function (Eq. 2) can thus be defined on each partition, which denotes the number of tuples that can be skipped for processing all queries in  $Q$ , if such a partition is formed. Here,  $|p_i|$  denotes the number of tuples in partition  $p_i$ , and  $Q_i$  denotes the set of queries that can be processed without accessing  $p_i$ .

$$\text{Cost}(p_i) = |Q_i| |p_i|, \text{ where } Q_i \subseteq Q \quad (2)$$

The MaxSkip partitioning problem aims to find the optimal partitions  $P_{opt}$  maximizing the total number of tuples that can be skipped when executing  $Q$ , i.e.,  $P_{opt} = \arg \max_P \sum_{p_i \in P} \text{Cost}(p_i)$ .

We map one instance of the MaxSkip partitioning problem to an instance of our optimal partitioning problem as below: for each query  $q_w$  in  $Q$ , we create a keyword  $d_w$  and form a SKR query  $q = \{q.L, q.d_w\}$  where  $q.L$  is the MBR of the entire space. For each tuple  $t_m$ , we create a geo-textual object  $o_m$  such that its location is in  $L$ , and its keywords correspond to the queries it can satisfy in  $Q$ .

Given this mapping, in the MaxSkip partitioning problem, for a partition  $p_i \in P$ , if a set of queries  $Q_i \subseteq Q$  can be skipped when processing  $p_i$ , we can get a cluster  $c_i$  in our problem and a set of SKR queries  $R_i \in W$  that are irrelevant to  $c_i$  (since no geo-textual objects in  $c_i$  contains a keyword in  $R_i$ ). Hence, if we could find an optimal partition that maximize the total number of tuples when running queries in  $W$ , it is equivalent that we can find an optimal partitioning method that minimizes the cost in our problem. Since the mapping is of linear time, we complete the proof.  $\square$

## 4.3 A Heuristic Partition Algorithm

As pointed out by Christoforaki et al. [13], a query region is usually much smaller than the data space such that many data objects are not queried by the workload  $W$ . Hence, to fully utilize the query workload for partitioning, a data based partitioning method is not suitable to solve our problem. Instead, we employ a space-disjoint partitioning approach and propose a heuristic partition algorithm.

Our index aims to learn splitting the spatial data space along different dimensions and coordinate values. Our partition algorithm starts by initializing one single partition that covers the full data space (which corresponds to a cluster that contains the full dataset). At this point, each query

contributes the same  $w_1 + |D| \cdot w_2$  cost to the overall cost of the query workload  $W$ . Then, we find a split dimension  $d_s$  and a split value  $v_s$  that yield the largest reduction in the query cost. We use the resulting  $d_s$  and  $v_s$  to split the data space into two sub-spaces and update the total query cost. For each sub-space, we repeat the splitting process recursively until the total query cost cannot be reduced or some pre-defined conditions, e.g., a minimum number of queries intersecting with the sub-space, are met. When the algorithm terminates, we use the MBR of the data objects in each resultant sub-space as a bottom cluster in WISK.

**4.3.1 Learning the Split Dimension and Value.** A naïve method to find the value to make a split uses a brute-force search. Let  $V_d$  ( $d \in \{x, y\}$ ) be a sorted list of distinct object coordinate values along dimension  $d$  in the current (sub-)space to be partitioned. Except for the first and the last values, every value in  $V_d$  can be used to split the space into two sub-spaces. Examining all  $|V_d|$  values takes  $O(f \cdot (|V_x| + |V_y|))$  where  $f$  denotes the time cost to split on a value and run queries based on such a splitting. This approach becomes impractical for large datasets with a large value of  $|V_x| + |V_y|$ .

Motivated by the recent success of machine learning in solving complex problems [4, 49], we propose a learning-based method to predict the query costs given a split dimension and a split value, such that the optimal split can be approximated by minimizing the predicted query cost with high efficiency. At the core of the query cost prediction problem of a split is to (1) predict the number of resultant sub-spaces overlapping with the query, and (2) predict the number of objects that contain any of the query keywords and reside in the resultant sub-spaces.

To address the first prediction problem, we use the indicator function [33] to denote whether a sub-space overlaps with the query region. For example, let  $[q_{x_b}, q_{x_u}]$  be the  $x$  range of query  $q$  and  $p_x$  be a split value along dimension  $x$ . The indicator functions  $\mathbb{1}(p_x \geq q_{x_b})$  and  $\mathbb{1}(p_x < q_{x_u})$  are used to decide whether  $q$  intersects with the resultant left and right sub-spaces, respectively. If a sub-space has an indicator function value of 1, we need to further predict the number of query result objects within the sub-space. Otherwise, we can ignore the sub-space when computing the query cost. The indicator function is not differentiable, and machine learning methods such as gradient descent cannot be applied to solve a split value optimization problem formulated by such functions. As such, we use the sigmoid function [24],  $\sigma(\beta x)$  with  $\beta = 3$ , to approximate the indicator function as does in prior work [8, 9], e.g.,  $\mathbb{1}(p_x \geq q_{x_b}) = \mathbb{1}(p_x - q_{x_b} \geq 0) \approx \sigma(3(p_x - q_{x_b}))$ .

To address the second prediction problem, we follow the idea in recent studies [36, 37, 45] that learn the Cumulative Distribution Function (CDF) to estimate the density of objects in a data space. Our goal is to learn the joint CDF  $F_{X,Y}(x, y)$  of two variables  $X$  and  $Y$ , corresponding to the spatial coordinates in two dimensions. The learned CDF can quickly estimate the number of objects in a rectangular region, i.e., a sub-space. To accelerate the CDF learning, we assume that  $X$  and  $Y$  are independent, following a previous study [45]. Thus, we can decompose the joint CDF into the product of two marginal CDFs,  $F_X(x)$  and  $F_Y(y)$ , as shown in Eq. 3.

$$F_{X,Y}(x, y) = P(X \leq x, Y \leq y) = F_X(x)F_Y(y) \quad (3)$$

For ease of presentation, we use  $F(x)$  and  $F(y)$  to denote the marginal CDFs of  $X$  and  $Y$  in the rest of the paper, respectively.

**LEMMA 4.2.** *Given a two-dimensional object  $(x, y)$  and a rectangular region  $[(x_b, y_b), (x_u, y_u)]$  where  $(x_b, y_b)$  and  $(x_u, y_u)$  denote the bottom-left and the upper-right points of the rectangular region, respectively, the probability of an object residing in the area is:*

$$P(x_b \leq x \leq x_u, y_b \leq y \leq y_u) = (F(x_u) - F(x_b))(F(y_u) - F(y_b))$$



PROOF. According to the definition of CDF, we have  $P(x_b \leq x \leq x_u, y_b \leq y \leq y_u) = F(x_u, y_u) - F(x_b, y_u) - F(x_u, y_b) + F(x_b, y_b)$ . Due to the independence assumption, we can decompose each joint CDF based on Eq. 3, and obtain the equation in Lemma 4.2.  $\square$

The CDF in Eq. 3 only estimates the spatial density of objects without considering the keyword distribution. To solve this issue, we learn the marginal CDFs, i.e.,  $F_k(x)$  and  $F_k(y)$ , for each keyword  $k$ . The choice of CDF models will be detailed in Section 6.

With the CDF models and the sigmoid functions, we formulate the cost for processing a query  $q$  with region  $[(x_b, y_b), (x_u, y_u)]$  after splitting on dimensions  $x$  or  $y$  in Eq. 4.

$$\begin{aligned} L_q(x) &= \sigma(3(x - x_b)) |O_1| + \sigma(3(x_u - x)) |O_2| \\ L_q(y) &= \sigma(3(y - y_b)) |O_1| + \sigma(3(y_u - y)) |O_2| \end{aligned} \quad (4)$$

where  $|O_1|$  and  $|O_2|$  denote the number of objects containing the query keywords in the two resulting sub-spaces, respectively, which are estimated through the learned keyword-based marginal CDF models. The sigmoid functions (e.g.,  $\sigma(3(x - x_b))$  and  $\sigma(3(x_u - x))$ ) predicts whether the query intersects the two resultant sub-spaces, respectively. We apply stochastic gradient descent (SGD) to minimize  $L_q(x)$  and  $L_q(y)$  using the query workload as the training data.

**4.3.2 Bottom Cluster Generation.** When splitting a data space, there are both profit and loss in the query costs. The profit is gained by the reduced number of objects to be checked while the loss reflects an increased number of sub-spaces to be checked. In Example 4.1, the profit and loss are equal to  $2w_2$  and  $2w_1$ , respectively. The difference between the profit and the loss determines whether a split is needed, and where the split should be made.

Algorithm 2 summarizes our bottom cluster generation algorithm. The algorithm takes the query workload  $W$  and the data space  $S$  enclosing all geo-textual objects as the input, and it aims to return a set of clusters that minimize the cost of executing all the queries in  $W$ . The algorithm maintains a priority queue  $Q$  of sub-spaces to be examined, which are prioritized by their numbers of intersecting queries. At the start,  $Q$  contains only the input data space  $S$  (lines 1 and 2). Then, we iterate through the sub-spaces in  $Q$ . Let the current sub-space to be split be  $s$ . We set the initial object checking the cost of  $s$  to be  $|O_s| \cdot |W_s| \cdot w_2$  where  $|O_s|$  and  $|W_s|$  denote the number of objects in  $s$  and the number of queries intersecting with  $s$ , respectively (lines 5 and 6). Then, we find the optimal split along both  $x$ - and  $y$ -dimensions, respectively (lines 7 and 8), and we use the one with a smaller object checking cost as our candidate split (line 9). If the reduction in the object checking cost from  $C_s$  outweighs the increase in cluster checking cost, i.e.,  $w_1 \cdot |W|$  (every split adds a cluster to be checked against  $|W|$  queries), we execute the split and enqueue the resultant sub-spaces (lines 10 to 13). Otherwise,  $s$  is finalized, and we generate the MBR for the data objects in  $s$  and use it as a bottom cluster (lines 14 to 16). The process terminates when  $Q$  becomes empty (line 4).

When finding the optimal splitting value along a dimension (lines 18 to 24), we apply SGD [53] to minimize Eq. 4 (line 21). Here, we use a map structure *opt* to record the new object checking cost, the dimension, and the value of a learned optimal split.

The time complexity of each iteration in Algorithm 2 is  $\mathcal{O}(h \cdot (E_x + E_y))$  where  $h$  and  $E_d, d \in \{x, y\}$  denote the time complexity of SGD per iteration and the number of epoches respectively. Recall that the time complexity of the brute-force algorithm is  $\mathcal{O}(f \cdot (|V_x| + |V_y|))$ . We note that  $f$  is larger than  $h$  because our heuristic algorithm does not need to run a split to calculate a query cost (while the brute-force algorithm does).  $E_x$  and  $E_y$  depend on the algorithm configurations, such as the learning rate and the number of model parameters. They are usually much smaller than  $V_x$  and  $V_y$ , respectively. Therefore, the time complexity of our heuristic algorithm is lower than that of the brute-force algorithm.

---

**Algorithm 2: Bottom Clusters Generation**


---

**Input:**  $W$ , the query workload;  $S$ , the data space

**Output:**  $G$ , the set of clusters

```

1  $Q \leftarrow \text{NewPriorityQueue}();$ 
2  $Q.\text{Enqueue}(S);$ 
3  $G \leftarrow \emptyset;$ 
4 while  $Q$  is not empty do
5    $s \leftarrow Q.\text{Dequeue}();$ 
6    $C_s \leftarrow \text{InitializeObjectCheckingCost}(s);$ 
7    $\text{opt}_x \leftarrow \text{FindOptimalPartition}(s, x);$ 
8    $\text{opt}_y \leftarrow \text{FindOptimalPartition}(s, y);$ 
9    $\text{best} \leftarrow \text{opt}_x$  if  $\text{opt}_x.\text{cost} \leq \text{opt}_y.\text{cost}$  else  $\text{opt}_y;$ 
10  if  $C_s - w_2 \cdot \text{best}.\text{cost} > w_1 \cdot |W|$  then
11     $s_1, s_2 \leftarrow \text{GenerateSubSpace}(\text{best}.\text{dim}, \text{best}.\text{val});$ 
12     $Q.\text{Enqueue}(s_1);$ 
13     $Q.\text{Enqueue}(s_2);$ 
14  else
15     $c \leftarrow \text{GenerateMBR}(s);$ 
16     $G.\text{add}(c);$ 
17 return  $G;$ 

18 Function  $\text{FindOptimalPartition}(s, d):$ 
19    $\text{opt}.\text{dim} \leftarrow d;$                                /* a map structure to record optimal split result */
20    $\text{cost}, \text{val} \leftarrow \text{SGDLearn}(s.\text{queries});$        /* SGDLearn() returns the optimal cost and split value */
21    $\text{opt}.\text{cost} \leftarrow \text{cost};$ 
22    $\text{opt}.\text{val} \leftarrow \text{val};$ 
23   return  $\text{opt};$ 

```

---

## 5 BOTTOM-UP PACKING

The bottom clusters generated from Section 4 can be used as a flat and coarse-grained index. To further improve the pruning power of our index, we build a hierarchical structure over the clusters.

### 5.1 Design Considerations

As shown in Section 3, when executing a query with a hierarchical index, we traverse all qualified nodes until reaching the leaf nodes. An internal node and its descendants can be pruned if it does not intersect with the query or include any query keyword. We build our hierarchical index level by level, i.e., recursively packing the clusters to maximize the reduction in the pruning cost at each level. Here we omit the object checking costs as they are only triggered on the bottom clusters.

**5.1.1 Optimization Goal.** The query time spent on node pruning can directly reflect the pruning capability of a hierarchical index. Measuring the query time, however, needs to run all queries in the query workload on an existing index, which is not suitable to be used as an optimization metric of our bottom-up packing problem. We observe that the pruning time cost is proportional to the number of accessed nodes for the workload such that it can be used to evaluate the pruning capability. To adopt this criterion, we associate each bottom cluster  $c_i$  with a query label set denoted by  $c_i.l$ . If a cluster  $c_i$  intersects with a training query  $q_j$  and its textual document includes any keyword of  $q_j$ , we add  $q_j$  to the query label set of this cluster, that is,  $c_i.l = \{q_j\}$ . During packing, the labels of a node in an upper level (an “upper node” for short hereafter) can be easily generated by merging all labels of its sub-tree.

**5.1.2 Bottom-up Packing Problem.** Next, we define the bottom-up packing problem to minimize the number of accessed nodes.

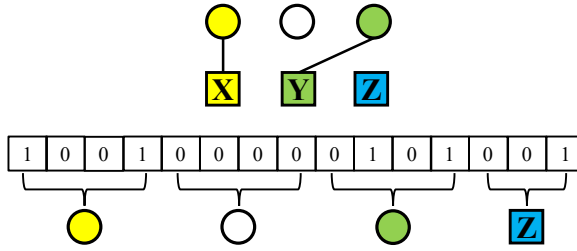


Fig. 6. An example of the state representation

**PROBLEM 2 (BOTTOM-UP PACKING).** *Given a query workload  $W$  and the set of bottom clusters  $G$ , the bottom-up packing process aims to generate a hierarchical index  $I$  that minimizes the number of accessed nodes to process the queries in  $W$ .*

Given the leaf nodes, i.e., bottom clusters, we can build a hierarchical index using techniques from traditional indexes such as the CDIR-Tree. However, those techniques only consider the underlying data distribution, which might lead to worse performance as shown in the later experiment.

To address these issues, and motivated by the strong performance of query-aware structures learned by reinforcement learning (RL), we propose an RL-based algorithm to learn a packing. We construct our index level by level with a bottom-up packing process, and we model the packing problem at each level as a sequential decision-making process, i.e., a Markov decision process, which makes it solvable by RL. To pack each level, the nodes from a lower level to be packed (“bottom nodes” hereafter) are processed sequentially, and we find an upper node to host each bottom node until there are no more bottom nodes. After the packing process of one level stops, the non-empty upper nodes become the new bottom nodes to be packed for the next level.

## 5.2 Packing with Reinforcement Learning

We propose an RL-based packing algorithm following the idea of the Deep-Q-Network (DQN) [44] to learn the optimal policy (i.e., a packing strategy) for solving the packing problem (Problem 2). To form a tree structure, we require that the number of upper nodes does not exceed that of the bottom nodes.

There are two main challenges in our packing problem.

- (1) To use a neural network to estimate an expected reward (e.g., the reduction in the number of node accesses), the states (e.g., the relation of two levels resulting from a packing decision) need to be represented by a fixed-length vector. However, there are many different possibilities of bottom nodes, and it is challenging to generate such a vector to encode the current packing of bottom nodes effectively.
- (2) Every time a node is added to the structure, it may lead to a reduced reward (i.e., more node accesses). However, it is necessary to add nodes to the structure continuously such that the structure can be built up. How to adapt the cost model for this case is another challenge.

To address these challenges, we formulate an MDP process for our packing problem as follows: **States.** A state needs to capture the status of a (partially packed) level in an index structure. As mentioned above, the number of bottom nodes bounds that of the upper nodes. Hence, we initialize  $N$  empty upper nodes given  $N$  bottom nodes. Consider  $m$  queries are used in the learning process. Each of the  $N$  upper nodes to be constructed takes an  $(m + 1)$ -dimensional vector representation. The first  $m$  dimensions denote whether the node is labeled by each of the  $m$  queries, and the last dimension is a count on the number of bottom nodes to be connected to this node. The  $N$  upper nodes together form an  $(m + 1) \cdot N$ -dimensional vector. We further append  $m$  dimensions to the

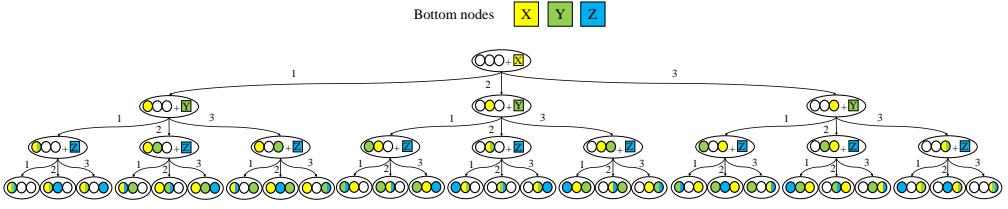


Fig. 7. An example of MDP formulation for Problem 2

vector to represent the query label of the next bottom node to be connected to (i.e., packed into) one of the upper nodes. Overall, these form an  $((m + 1) \cdot N + m)$ -dimensional vector representing a state. Figure 6 shows an example, assuming  $m = 3$  queries and  $N = 3$  bottom nodes ( $X$ ,  $Y$ , and  $Z$ ). The circles denote upper nodes, and the colors denote different query labels.

**Actions.** An action adds a bottom node to an upper node. To make the action space and the state representation consistent, we define the action space  $A = \{1, 2, \dots, N\}$ , where action  $a = i$  denotes packing the next bottom node into the  $i$ -th upper node.

**Transition.** Given a state and an action, the agent transits to a new state by packing a bottom node into the chosen upper node and moving on to the next bottom node. The agent reaches a terminal state when there are no more bottom nodes to be packed.

**Reward.** A larger reward represents a packing with better quality. Since we aim to reduce the number of node accesses when executing the query workload, the reward signal should reflect the expected number of node accesses before and after taking an action.

We propose to use the average number of node accesses per query to formulate the rewards since the total number of node accesses grows monotonically as more bottom nodes are added to the consideration, which will lead to constant negative rewards.

$$r = N_a - N'_a \quad (5)$$

The reward function is formulated as Eq. 5, where  $N_a$  ( $N'_a$ ) denotes the average number of node accesses before (after) action  $a$  is taken. The agent chooses the action that maximizes the reward during exploitation. Additionally, we observe a positive correlation between the sum of rewards and the reduction in the average number of node accesses after packing all bottom nodes. Let  $N_a^*$  be the average number of node accesses of packing the last bottom node. As  $N_a$  in each iteration is identical to  $N'_a$  of the last iteration, the sum of rewards after packing all  $N$  bottom nodes is equal to  $1 - N_a^*$ , and it is positively correlated to  $N + 1 - N_a^*$ . Note that the number of node accesses is equal to  $N + 1$  before creating the upper nodes. Thus, if the sum of the rewards at a level is not larger than  $-N$ , the bottom-up packing process will be terminated.

**Example 5.2:** Figure 7 presents an example of the MDP for the bottom-up packing problem. Same as in Figure 6, the colors represent different query labels. Here, we only show state transitions with nonzero probabilities, and we have omitted the rewards to avoid clutter. The ellipse nodes and edges represent states and actions, respectively. Since there are 3 bottom nodes (rectangle), we initialize 3 upper nodes (circle), and the bottom nodes are to be packed sequentially. When no incoming bottom node is to be inserted at one level, i.e. the leaf node in Figure 7, we reach the terminal states at this level and move to the upper level.

### 5.3 Training

Recall that Q-learning is a commonly used RL algorithm as introduced in Section 2.2. We train a deep Q-network (DQN) [44] to project the high-dimensional state and action spaces to low-dimension spaces using neural networks and efficiently predict the value of the Q-function  $Q(s, a)$ . In our model, we adopt the deep Q-learning with a technique known as experience replay where we

**Algorithm 3:** DQN Learning for Node Packing

---

**Input:**  $G$ , the bottom nodes with query labels;  $M$ , replay memory;  $E$ , the number of epochs  
**Output:**  $Q(s, a; \theta)$ , action-value function

```

1 Initialize  $Q(s, a; \theta), \hat{Q}(s', a', \theta^-)$ ;
2 for  $epoch \in [1, E]$  do
3    $G_u \leftarrow \text{NewList}(); M \leftarrow \text{NewList}();$ 
4   for  $c_i \in G$  do
5     Update  $s$  using  $G_u$  and  $c_i$ ;
6     Compute the average number of node accesses  $N_a$  according to  $G_u.l$ ;
7     Choose  $a$  by the  $\epsilon$ -greedy method;
8     Pack  $c_i$  into  $G_u[a]$  and generate the new state  $s'$ ;
9     Compute  $N'_a$  according to the new  $G_u.l$ ;
10    Compute reward  $r$  based on Eq. 5;
11    Store transition  $(s, a, r, s')$  into  $M$ ;
12    Draw a batch of samples from  $M$  and perform a gradient step based on Eq. 6;
13    Update  $\hat{Q}(\cdot; \theta^-)$  with  $Q(\cdot; \theta)$  softly based on Eq. 7 after every  $C$  steps;
14 return  $Q(s, a; \theta)$ ;
```

---

store the agent's experience  $e_t = (s_t, a_t, r_t, s_{t+1})$  at each time-step  $t$ . We implement two networks, a policy network  $Q$  and a target network  $\hat{Q}$  separately, which has been shown to be more stable than using only one network as done in the standard Q-learning [44].

Given a batch of transitions  $(s, a, r, s')$ , the policy network parameters  $\theta$  are updated with a gradient descent step by minimizing the mean square error (MSE) loss as shown in Eq. 6, where  $\gamma \in (0, 1)$  denotes a discount factor determining the importance of future rewards, and  $\theta^-$  are the parameters of the target network.

$$L(\theta) = \sum_{s,a,r,s'} \left( r + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \quad (6)$$

Note that the target network parameters  $\theta^-$  are only synchronized with the policy network parameters  $\theta$  every  $T$  steps and are held fixed between weight updates. However, directly copying the weights has been shown to be unstable due to noise and outliers. Inspired by prior works [34, 39], we apply the soft update (Eq. 7) to the target network. The weights of the target network are updated by interpolating between the weights of the target network and those of the policy network through a fixed ratio  $\tau = 0.001$  [39].

$$\theta^- = \tau\theta + (1 - \tau)\theta^-, \tau \ll 1 \quad (7)$$

We present the learning process in Algorithm 3. We first initialize the policy network and the target network with the same random parameters (line 1). In each epoch, we reset the replay memory  $M$  and the set of upper nodes  $G_u$  (line 3). Then, the learning process sequentially packs the bottom nodes to the upper nodes (lines 4 to 13). For every incoming bottom node  $c_i$ , we generate the state by combining  $G_u$  and  $c_i$  (line 5). We compute the average number of node accesses based on the query labels of the current upper nodes (line 6). To balance between RL exploration and exploitation, we use the  $\epsilon$ -greedy algorithm [57] to choose a random action with probability  $\epsilon$  (i.e., exploration) or the action that maximizes the action-value function of the policy network (i.e., exploitation) (line 7). After  $c_i$  is packed, we update the state representation and compute the average value again (lines 8 and 9). Then, we compute the reward and store this transition in the replay memory (lines 10 and 11). To train the DQN, we draw a batch of transitions to train the policy network (line 12) and periodically copy the policy network parameters to the target network (line 13). Finally, we use the learned action-value function  $Q(s, a; \theta)$  to pack the nodes.

## 6 DESIGN OPTIMIZATIONS

**Choice of CDF models.** For our heuristic partition algorithm, the number of objects with each keyword is approximated by a model that learns the corresponding CDF. Prior works [36, 38] have used the neural network (NN) to learn the CDF. However, learning an NN for each query keyword may lead to a large number of NNs to be learned and hence high preparation costs.

We observe that the query time of WISK is more sensitive to high-frequency keywords. To decrease preparation costs, we divide keywords into three classes based on their frequency: low ( $\leq 0.001\%$ ), medium ( $0.001 - 0.1\%$ ), and high ( $\geq 0.1\%$ ). Previous studies [64, 72] have shown that more resources should be allocated to records with more frequent elements to get better prediction accuracy. When calculating the query cost, low-frequency keywords are ignored as they have little impact on the query time. We adopt Gaussian functions to approximate the data distribution of each medium-frequency keyword and learn an NN to approximate the CDF of each rest keyword. Our empirical results show that such a strategy balances the preparation costs and the query time.

**Correlation between keywords.** In Section 4, we consider each keyword independently when approximating  $|O_s|$  ( $s \in \{1, 2\}$ ) in Eq. 4. This independence assumption impacts the performance of the heuristic partitioning algorithm when a query has more than one keyword, e.g., if an object contains  $k$  query keywords of a query, this object will be counted  $k$  times when predicting the number of objects in a sub-space, leading to inaccurate query cost prediction.

To solve this issue, we exploit *frequent itemset mining* to discover all frequent keyword sets and extract associations among the given set of keywords [1, 2, 25, 59]. We apply a classic algorithm, FP-Tree [25], to find frequent keyword sets from the underlying data. Then, we learn a CDF model of objects containing all keywords in one frequent keyword set and use the learned model to predict the number of objects with the set of query keywords more accurately.

**Action mask in RL.** When the packing of a level starts, the upper nodes are all empty. To choose the upper node to insert for the first bottom node, we observe that actions  $a = i$  ( $i > 1$ ) are all equivalent to  $a = 1$ . We call these actions *duplicated actions*. Duplicated actions exist when more than one upper nodes are empty. As observed in Figure 7, the actions of adding the bottom node to any of these empty upper nodes are equivalent. Such duplicated actions make the exploration inefficient, leading to slow convergence [23].

Thus, motivated by a prior study [76], another use of the environment is to generate an action mask based on the current state to hide the duplicated actions from the agent. In the example above, before inserting the first bottom node, the action mask generated by the environment makes the agent only chooses action  $a = 1$ .

**Training time acceleration.** WISK has two steps: finding the bottom clusters and packing the bottom clusters through RL. To reduce the training time of WISK, we design acceleration techniques for both steps. The first technique is to use sampled training queries, following a previous work [45]. We use stratified sampling [5] to obtain query samples that can better represent the distribution of the original workload. The second technique groups the bottom clusters using a clustering algorithm to reduce the number of bottom clusters to be packed in the bottom-up packing step. We utilize the spectral clustering [46] with the coordinates of the bottom left and top right points of each bottom cluster as features.

## 7 EXPERIMENTS

### 7.1 Implementation and Setup

**Implementation.** The learning process of CDF NN models, Algorithm 2, and Algorithm 3 are implemented with PyTorch [47]. The performance evaluation of all index structures is implemented in C++ and compiled using GCC 9.3 with -O3 flag. In the process of generating the bottom clusters,

we empirically set 0.1 and 1 to the weights of stage 1 and stage 2, i.e.,  $w_1$  and  $w_2$ , respectively. The CDF network consists of 4 layers, and each hidden layer has 16 units. We use ReLU as the activation function of the hidden layer. The output of the CDF is activated by a sigmoid function. When packing the bottom clusters, we follow the original implementation of DQN [44]. The neural network consists of 3 layers, and each hidden layer has 64 units. We set the capacity of experience replay to 256, and the discount factor is set to 0.99. For  $\epsilon$ -greedy algorithm, the initial value of  $\epsilon$  is set to 1, and the value decreases with more learning steps, which balances exploration and exploitation well.

**Environment.** We run single-threaded experiments in the main memory on an Ubuntu machine with Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz, 128GB RAM, and a 500 GB SSD disk. Besides, we train our CDF models on an Ubuntu machine with Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz, 256GB RAM, and RTX 2080 Ti GPU.

**Baselines.** We compare WISK with four SOTA conventional indexes, i.e., **CDIR-Tree** [15], **SFC-Quad** [13], **ST2I** [27], and **ST2D** [58]. We implement these indexes using the default parameter values reported in their original papers. Note that ST2D is only evaluated on **FS** by setting the similarity threshold to 0 since it is only suitable for the case that containing a few distinct keywords (a few hundreds) because of the textual clustering.

We also integrate a learned spatial index with a textual index loosely, following traditional spatial keyword indexes. This results in a *learned spatial-first index (SFI)* and a *textual-first index (TFI)*. SFI attaches an inverted file for keywords indexing to each leaf node of a learned spatial index, while TFI uses an inverted file as its top-level index and creates a learned spatial index for the objects containing the same keyword. It has been shown that textual-first indexes outperform their spatial-first counterparts [61, 79]. Therefore, we only report results for TFI in our experiments. LISA [38] is used as the learned spatial index since it returns the exact results. We further extend a learned multi-dimensional index, i.e., Flood [45]. We build an inverted file for each grid cell in Flood and also improve its cost function for building the grid index by incorporating the textual information, utilizing our CDF models on the geo-textual data, following the method presented in Section 4. We denote this index by **Flood-T**. It splits the data along *only one dimension* in the 2D geographical space, which limits its capability to capture the complex data distribution. We also compare with **LSTI** [19], the latest index to support spatial keyword queries. This method maps the data into one dimension using a Z-order curve based on the spatial coordinates and builds a RadixSpline index [32] using the mapped values. Then, an inverted file is created for each spline point by scanning the dataset again.

## 7.2 Datasets and Workloads

We use three real-world datasets in Table 1. The **FS** dataset [71] consists of global-scale check-in records of Foursquare (<https://foursquare.com/>) from Apr. 2012 to Jan. 2014. A check-in data has a spatial location and its category. The **SP** dataset includes recreational and sports areas extracted from OpenStreetMap (<https://www.openstreetmap.org>). We use the center of each area and the original description as the spatial location and keywords, respectively. The **BPD** dataset contains global POIs published by the SLIPO project [48] (<http://slipo.eu/>). The **OSM** dataset contains 100M POIs extracted from OpenStreetMap, which is published in UCR STAR [22]. Each POI has a point location, and its keywords include all related information such as street and category. As there is no public real-world query workload for the geo-textual datasets, we generate the queries by following previous works [10, 23, 28, 63, 65]. Specifically, to generate a query, we first sample an object in the dataset, and then generate a bounding rectangular area with the location of this object being its center. Inspired by previous works [23, 63], we use four methods to generate the centers: (i) **UNI**, where centers are uniformly sampled from the dataset. (ii) **LAP**, where centers are

Table 1. Dataset Statistics

Property	FS	SP	BPD	OSM
Number of data objects	3M	4M	25M	100M
Number of distinct keywords	462	1M	24M	447M
Total number of keywords	6M	11M	116M	478M

Table 2. Parameters and their settings

Parameter	Setting
Query distribution	UNI LAP GAU <u>MIX</u>
Query region size (%)	0.005 0.01 <b>0.05</b> 0.1 0.5 1
Number of query keywords	1 3 <u>5</u> 7 9

sampled from the Laplace distribution [35]. We set the location and scale parameters, i.e.,  $\mu$  and  $b$ , to  $|D|/2$  and  $|D|/10$  respectively, where  $D$  is the object set. (iii) **GAU**, where centers are sampled from a Gaussian distribution ( $\mu = |D|/2$ ,  $\sigma = 100$ ). (iv) **MIX**, composing of the centers generated from the (i) and (ii) in equal proportions. Finally, we associate keywords for the queries following prior works [10, 65]. If the number of query keywords is less than the number of keywords of the center, we choose the query keywords from the sampled object. Otherwise, we randomly choose the remaining keywords from the global keyword set.

To evaluate the performance of indexes in different scenarios, we generate query sets with different numbers of keywords and query sizes. Table 2 summarizes parameters, where default values are in bold and underlined. We generate 2000 queries under each setting, in which 1000 queries are utilized to test the performance of all the indexes, and others are used to train learned indexes.

### 7.3 Query Time Evaluation

To evaluate the query time, we execute testing queries 100 times and report the average cost of the queries in each query set.

**7.3.1 Effect of query distribution.** In this experiment, we fix other settings except for the query distribution and show the results on all datasets in Figure 8. Clearly, conventional indexes (SFC-Quad, ST2I, and CDIR-Tree) perform worse on the skewed workload since they do not use the query characteristics when constructing the index. For the learned indexes, TFI performs even worse than the conventional indexes since it only loosely combines a learned spatial index with a textual index. Flood-T shows a slight fluctuation in its performance since it learns from the underlying data and the query workload simultaneously, but in the geo-textual scenario, it only splits along one dimension, making it incompatible with the skewed workload. Our WISK improves the partitioning and adopts RL to build a tree, so it is less sensitive to this alteration.

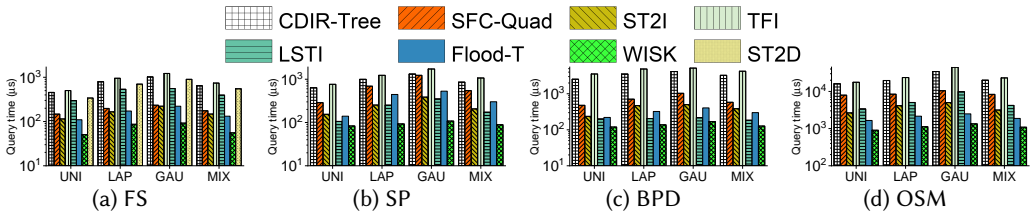


Fig. 8. Varying the distribution of the query workload

**7.3.2 Effect of query region size.** We show the performance of all indexes, by varying the query region size varies from 0.005% to 1% of the whole region in Figure 9. Again, WISK performs the best



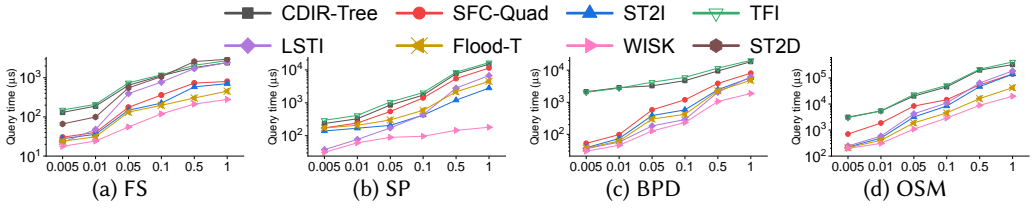


Fig. 9. Varying the query region size

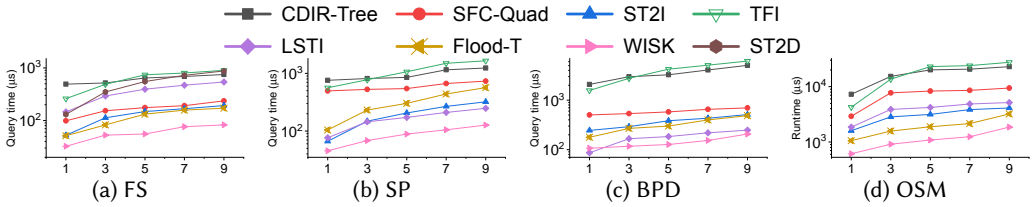


Fig. 10. Varying no. of query keywords

on four datasets. Besides, Flood-T performs slightly worse than ST2I on **SP**, even though it optimizes its layout by learning from the data and query workload. It is because it only splits the whole region along one dimension. Thus, we improve this process to generate the leaf nodes of WISK and also pack the bottom clusters into a hierarchical structure. The two techniques simultaneously result in the superiority of WISK over the other indexes.

**7.3.3 Effect of number of query keywords.** We evaluate the query sets with different numbers of keywords. Figure 10 shows that the query time of all indexes grows with the query keyword set size. The reason is that with the increase in the number of query keywords, more candidates need to be verified after the filtering step. Besides, WISK consistently outperforms other baseline indexes, and its cost grows much slower than those of others, e.g., the increased time of WISK on **BPD** is around 100  $\mu$ s while those of Flood-T and ST2I are both over 250  $\mu$ s. Hence, compared to other indexes, WISK is less sensitive to the number of query keywords.

**7.3.4 Scalability.** We generate five sub-datasets of **OSM** containing from 1 to 100 million objects and run experiments on these sub-datasets. We choose ST2I, LSTI, and Flood-T as our baselines. As shown in Figure 11, the query processing time increases with the size of the dataset, but WISK performs more stable.

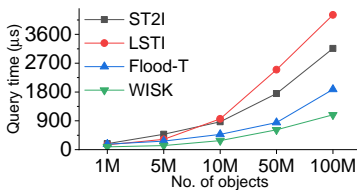


Fig. 11. Comparison of performance varying dataset size (num records)

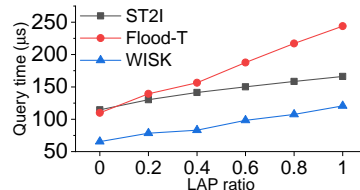


Fig. 12. Comparison of performance when changing query distribution

**7.3.5 Robustness.** We evaluate the performance of ST2I, Flood-T, and WISK when the query distribution changes on **FS**. We initially train Flood-T and WISK based on the query workload with **UNI** distribution. Then, we keep the index consistent and adjust the ratio of queries with **LAP** distribution from 0.2 to 1.0 in the testing query set. As shown in Figure 12, the performance of query-aware indexes becomes worse when query distribution is more different from the training

one. However, it can be seen that WISK is more robust than Flood-T due to its improved partitioning algorithm and the bottom-up packing process. Additionally, the query time of ST2I also increases since it ignores the query knowledge when building the index, but the fluctuation is less than the one of Flood-T.

## 7.4 Index Size & Construction

**7.4.1 Index Sizes.** Table 3 reports the index sizes. Overall, WISK costs less space than conventional indexes but is comparable to that of the best adapted learned indexes. In particular, the size of CDIR-Tree is larger than those of the others since each of its nodes has an inverted file. For query efficiency, we have not compressed SFC-Quad, which leads to a larger size. ST2I has the smallest size among the conventional indexes. Among the learned indexes, the sizes of TFI are the largest, as it uses inverted files. WISK takes more space than Flood-T on a small dataset since the number of its bottom clusters is similar to the number of the columns of Flood-T, but we build a hierarchical index. However, on larger datasets, WISK needs less space cost, since Flood-T splits more columns for better performance and builds inverted files for them.

Table 3. Index structure size

Index	FS	SP	BPD	OSM
CDIR-Tree	2002MB	3571MB	33.15GB	108.45GB
SFC-Quad	1406MB	2568MB	15.65GB	58.71GB
ST2I	761MB	1554MB	15.18GB	56.05GB
TFI	573MB	1423MB	8.86GB	32.05GB
LSTI	642MB	1073MB	8.85GB	8.09GB
Flood-T	400MB	937MB	7.15GB	27.94GB
WISK	483MB	980MB	7.02GB	25.78GB

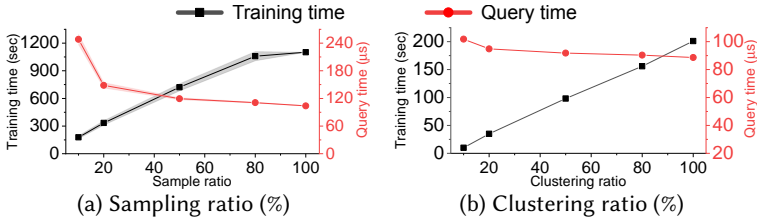
**7.4.2 Index Construction Time.** We compare the efficiency of index construction algorithms and report the results in Table 4. It takes the minimum time to build SFC-Quad and ST2I on small datasets. However, the time cost of ST2I significantly increases when the dataset becomes larger, since ST2I is built based on the set of converted points, and its time cost is positively correlated to the total number of keywords. CDIR-Tree takes the highest time cost because it inserts the objects sequentially.

Table 4. Index construction time

Index	FS	SP	BPD	OSM
CDIR-Tree	391 sec	490 sec	56.17 min	196.87 min
SFC-Quad	20 sec	30 sec	3.35 min	9.18 min
ST2I	19 sec	29 sec	6.55 min	26.23 min
TFI	125 sec	283 sec	33.75 min	143.07 min
LSTI	23 sec	32 sec	4.16 min	16.32 min
Flood-T	188 sec	974 sec	19.66 min	25.97 min
WISK (Accelerated)	131 sec	547 sec	12.18 min	17.43 min

For the learned indexes, we report the training time. LSTI takes the least time to build because it only needs to scan the whole dataset twice. The time cost of TFI increases significantly when there are more different keywords. For Flood-T and WISK, we report the average time as the time costs of query-aware learned indexes usually increase with more query keywords.

We designed two training time acceleration techniques as presented in Section 6. We report the training and query times of WISK with different sampling ratios in Figure 13a. The result of

Fig. 13. Training time and resulting query time on **SP**

each sampling ratio is an average of 10 runs. While the training time decreases by 72%, we do not observe a large drop in query performance with a sample of only 30% of the full query workload. We also observe that the standard deviation (represented by the width of the bands) of the training and query times of WISK is consistently small for all sampling ratios. This demonstrates that WISK has a stable performance using stratified sampling. We vary the clustering ratio, i.e., the number of groups obtained over the number of bottom clusters, to balance the training and querying time. Figure 13b shows that even when the number of bottom clusters decreases by 80%, the query time of WISK still only changes slightly. We set the sampling ratio to 30% and the clustering ratio to 20% and generate the **Accelerated WISK**. As shown in Table 4, WISK has longer training times than the other learned indexes, but the acceleration techniques can reduce index training time up to 4 times while the query time is only affected marginally.

## 7.5 Index Update

**7.5.1 Dynamic Query Workload Changes.** To update the index when query distribution changes, we can retrain WISK periodically following the former study [45]. We generate six workloads for **FS**. For each workload, we randomly select the query region size and the number of query keywords, and the query distribution adopts the default settings (**MIX**) and we randomly select the proportions of **UNI** and **LAP**. Each workload runs for 30 minutes and consists of 100 queries. As Figure 14 shows, at the start of each 30-minute period, i.e., a new query workload starts, retraining WISK is triggered, which happens in a separate thread and does not interrupt the query processing. While the index is being rebuilt, WISK runs the new queries on its old layout, which explains the jumps in the figure. The retraining lasts about 3 minutes, and then WISK switches to the new layout adapted to the new query workload. Thus, the query time drops back again.

To capture minor changes in query distribution when retraining, we propose to apply incremental updates to the original index, in parallel to the retraining process. We locate the bottom clusters that are affected by the new queries, re-partition these clusters if the query costs can be reduced using new queries, and then insert the new clusters back into the non-leaf nodes they previously belonged to. The incremental updates may also help reduce the query times, which explains the multiple drops (e.g., at 00:30 and 01:30).

Figure 14 also indicates the necessity of learning from the query workload. When a new workload arrives, the performance of WISK drops due to its outdated layout. Re-learning the layouts based on the new workload mitigates the impact of the changing query distribution. The other two indexes do not utilize the queries during construction, which leads to much worse performance than WISK (e.g. at 01:00, 02:00, and 02:30) on the skewed query workloads (i.e. high proportion of **LAP**).

**7.5.2 Data Insertion.** WISK also handles data insertion well. Given a new object  $o$ , we can traverse WISK to find the bottom cluster where  $o$  falls. Next, we update the inverted file or the bitmap of the affected nodes to obtain an updated index. This simple process, however, cannot guarantee an optimal layout because the bottom clusters might need to be split after the insertion. Thus, we buffer the inserted objects and retrain our index when the buffer is full.

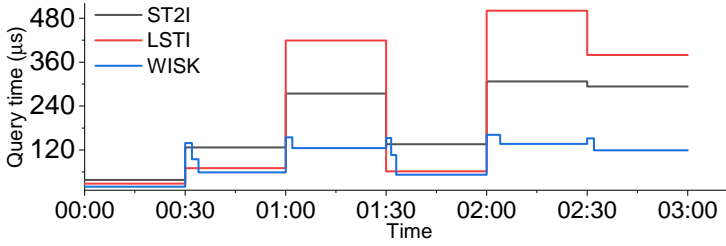


Fig. 14. Impact of dynamic workload changes

We set the buffer size at 100,000 (around 20MB) and run experiments to examine the impact of data insertions. We randomly select 500,000 objects from **FS** for the insertions. We insert 100,000 objects every 30 minutes. Figure 15 shows the performance of ST2I, LSTI, and WISK. We compare with WISK using the simple insertion process without retraining. It can be seen that the query time of all indexes increases when more objects are inserted. Between the two WISK variants, we see that the query time of WISK without retraining increases faster with more insertions, thus verifying the importance of retraining in improving the query time of WISK under dynamic data settings. We also observe that the retraining process takes only 1 to 2 minutes each time, since only the affected bottom clusters need to be split, and the RL-based packing can inherit knowledge from the previous training process, i.e., the unaffected bottom clusters are initially packed into the previous corresponding upper nodes.

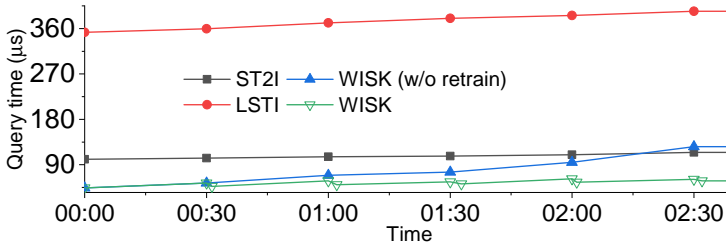


Fig. 15. Impact of data insertion

## 7.6 Ablation Study

**7.6.1 RL-based Packing.** We conduct an experiment to compare the cost at the leaf level and that at the non-leaf level. Figure 16 shows that the time at the leaf level dominates the query processing time, which occupies around 90% of the total cost, verifying the way to define cost function is reasonable. In Figure 17, we observe that packing our bottom clusters by directly using CDIR-Tree construction method might affect the query time because it may pack some leaf nodes intersecting with various queries.

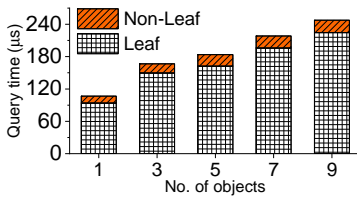


Fig. 16. Comparison of processing time

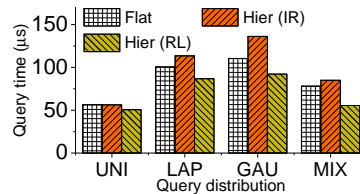


Fig. 17. Comparison of packing methods

We evaluate the effectiveness of the bottom-up construction process. As shown in Figure 18a, the improvement of the different number of keywords is similar. This is because the number of

query keywords has little effect on the number of bottom clusters. Thus, the improvement is stable using this RL-based grouping algorithm.

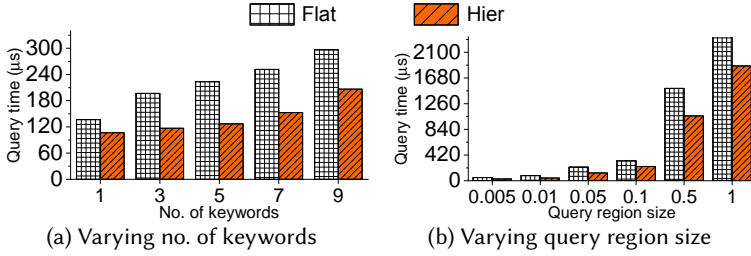


Fig. 18. Comparison of index layouts on BPD

It can be seen from Figure 18b that the improvement of hierarchical indexes becomes more significant for queries of larger region sizes. This is because queries of larger region size correspond to a wider space covered by these queries and more bottom nodes such that the bottom-up packing process can reduce more filtering cost. However, we also observe that the improvement becomes stable as the region size continues to get larger, as the query region covers most of the data space.

**7.6.2 CDF Model.** When generating bottom clusters, we use CDF models to estimate the number of objects sharing the same keywords inside a region. To reduce the parameters, we propose to use Gaussian functions and NNs for keywords with different frequencies. In Figure 19a, we compare our method with the settings in which only Gaussian models or NN models are used. Although the Gaussian-only method has the least training time, its estimation results are inaccurate, leading to much worse query time. In contrast, the NN-only method achieves the best query time, but it needs much more training time. In comparison, the proposed mixed method achieves similar query performance as the NN-only method without significantly increasing the training time.

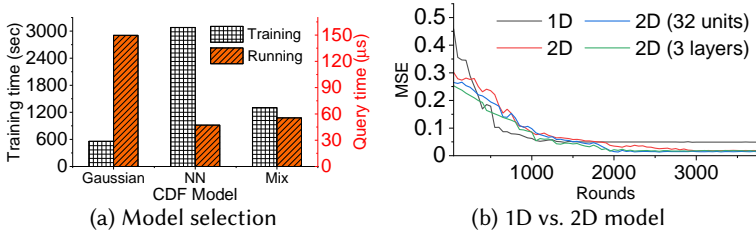


Fig. 19. Effect of different CDF settings

To speed up pre-processing, we assume that the two spatial dimensions are independent following the existing work [45]. We next study the impact of such an assumption.

We observe that keywords with higher frequency have a stronger impact on the query time, and thus they need a more accurate CDF estimation. We run experiments with a randomly selected high-frequency keyword on FS. We train two marginal (1D) models and a joint (2D) model for the selected keyword on the whole dataset. The 1D models and the 2D model all employ a neural network with 2 hidden layers and 16 hidden units. We also compare with two variants of the 2D model, one uses more hidden units (32 units) and the other uses more layers (3 layers). We randomly sample 1,000 rectangular query regions within the data space as testing data. For each query region, we compute the proportion of objects that fall within it as the ground truth. The product of the two 1D models and the output of the 2D models are used as the estimation results corresponding to 1D and 2D models, respectively. For every 50 training rounds, we calculate the mean squared error (MSE) between the estimations and the ground truth. Figure 19b shows that the 1D model

converges much faster while its final loss is comparable to those of the 2D models. These results justify the use of 1D CDFs in our method.

**7.6.3 Frequent Itemset (FI).** The FI mining extracts the frequent interrelation among keywords. In this experiment, the minimum support is set to 0.01‰ and the maximum size of target itemsets is equal to the number of query keywords. Figure 20 shows the effect of FI on the index construction by query efficiency on **FS** and **BPD**. We see that the FI mining improves the performance of WISK consistently when there is more than one query keyword. Without the FI mining component, we learn models of each keyword separately where redundancies occur when there is more than one keyword. We also observe that this adaptation is more beneficial given more query keywords. In general, more query keywords lead to a higher possibility of resulting in redundancy since the probability of an object including more than one query keyword increases.

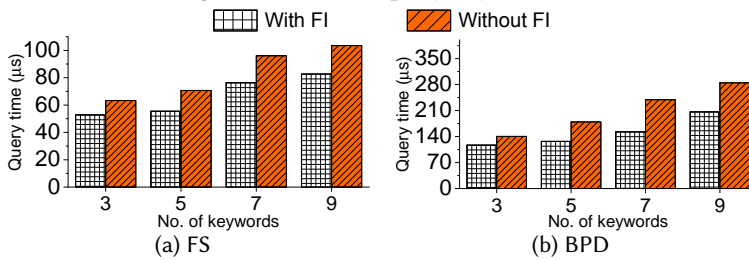


Fig. 20. Effect of the frequent itemset

Besides, the performance improvement on **BPD** is more obvious than that on **FS**. This is because the number of distinct keywords on **FS** is much less, making the number of frequent itemsets less than others. Additionally, the improvement becomes consistent when the number of query keywords is larger than a threshold. The reason is that each object includes finite keywords, and thus we cannot generate frequent itemsets with more keywords.

**7.6.4 Action Mask.** When using the RL framework, the environment applies the action mask to reduce the action space. Here, we evaluate its effectiveness in two aspects. We use the SmoothL1Loss with the sum reduction as the loss function in our RL framework. Figure 21a shows that the RL framework with the action mask can speed up the model convergence and reach a smaller loss. In Figure 21b, we sum the total rewards and the number of bottom nodes in each epoch to show the reduction in the average number of accessed nodes. The result shows that the pruning capability is always better when applying the action mask.

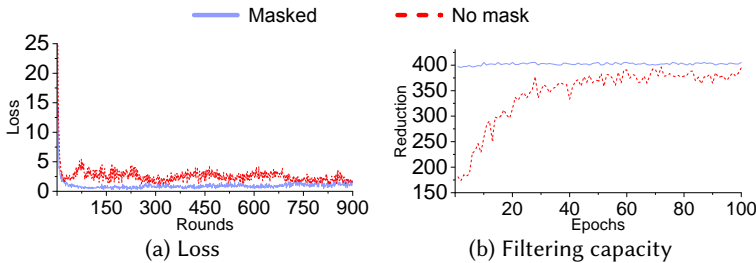


Fig. 21. Comparison of model convergence and reward

there are some fluctuations in the results as the RL agent learns from the feedback through trial-and-error interactions with the environment. RL balances the trade-off between exploration and exploitation, which reduces the fluctuation with more training epochs. These results confirm that the action mask helps to decrease the number of training epochs, leading to less training time.

## 7.7 Parameter Sensitivity Study

We further evaluate the sensitivity of WISK's training and query times to our key parameters. We show the results of varying the numbers of hidden units and layers in the neural networks in Figure 22a. Using more hidden units significantly increases the training time but only slightly improves the query time. Increasing the number of hidden layers shows a similar effect. Additionally, increasing the size of the structure requires larger memory space. Thus, we set the default hidden units and layers to 16 and 2.

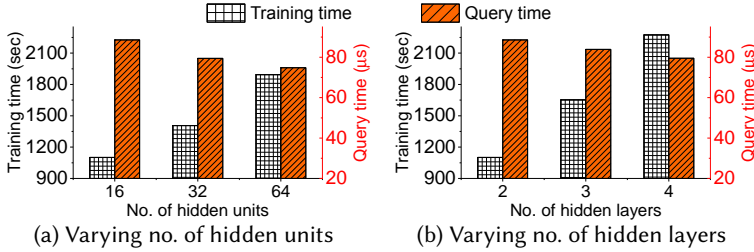


Fig. 22. Comparison of training time and query time

We also vary the capacity of experience replay and the discount factor. The results show that they have minor impacts on the query time and the RL convergence rate. We omit the details due to space limits. The performance of WISK is less sensitive to these hyper-parameters, and we can follow the existing work [44] to set them.

## 8 RELATED WORK

**Traditional geo-textual indexes.** In recent years, due to the popularity of SKR queries and their applicability in practical scenarios, a series of geo-textual indexes [11, 13, 26, 31, 58, 61, 79] have been proposed to support efficient SKR query processing.

The general idea of geo-textual indexes is to combine spatial and textual indexes to exploit their pruning capabilities based on the spatial and textual attributes of the data. Early works only loosely combine both types of indexes. For example, Vaid et al. [61] propose the first grid-based geo-textual indexes, i.e., the spatial primary index (ST) and the text primary index (TS), which are spatial-first and textual-first integrations, respectively. Parallel to this work, the  $R^*$ -tree-inverted file ( $R^*$ -IF) and the inverted file- $R^*$ -tree (IF- $R^*$ ) [79] combine the inverted file with the  $R^*$ -tree [3]. The loose integrated indexes has been shown to result in unsatisfactory query time in both the follow-up study [10] and our experiments.

Later works combine spatial and textual indexes more tightly such that they use both types of attributes of the data for search space pruning in parallel. For example, each grid cell in Spatial-Keyword Inverted File (SKIF) [31] is presented by an inverted list, and it works in the rectangular object scenario. Hariharan et al. [26] propose the Keyword- $R^*$ -tree ( $KR^*$ -tree). Each node of this index is associated with the set of keywords that appear in the sub-tree rooted at this node. Thus, each tree node can prune the search space with both a spatial region and a set of keywords at the same time.

The indexes above focused on SKR queries. There are also geo-textual indexes [15, 16, 27, 52, 68, 75], such as IR-Tree, developed for other types of spatial keyword queries. Some of these can be adapted to answer SKR queries. However, since they are not tailored for SKR queries, their query time is usually worse [10].

**Learned indexes.** Kraska et al. [36] propose the recursive model index (RMI), which leverages machine learning models to replace a traditional index over one-dimensional search keys. The motivation is that an index can be seen as a function mapping a search key to the storage position

of the corresponding record. Several follow-up studies propose learned indexes for one-dimensional data [17, 21, 69]. More details can be found in a benchmark study [42].

To handle multi-dimensional data, the Z-order model [62] extends RMI by utilizing a Z-order curve to map multi-dimensional search keys into one-dimensional keys. Since this index might lead to large and uneven gaps between the mapped keys of adjacent objects, RSMI [51] proposes a rank space-based technique, and it further proposes a hierarchical learned partitioning strategy for index learning over large spatial datasets. A parallel work, LISA [38], designs a grid-based index that supports data updates. The RLR-tree [23] uses machine learning techniques to build a better R-tree without the need to change the structure or query processing algorithms of the R-tree.

Although these indexes have shown performance gains by exploiting the data distribution, they have ignored the query workload in index construction. Several studies [18, 40, 45] take into account the query workload and propose to automatically optimize the index structure for a given data and query distribution.

**Reinforcement learning.** RL is often utilized in sequence generation applications, such as game playing [54], machine translation [30], and bin packing [78]. Recently, it has been adapted to solve database optimization problems, such as query optimization [76], index tuning [70], and trajectory simplification [66]. However, these problems are quite different from ours by definitions, and so are their state and reward formulations. Thus, our RL formulation requires new designs in the states and reward functions.

## 9 CONCLUSIONS AND FUTURE WORK

We proposed a hierarchical index named WISK for SKR queries, which is jointly optimized for a given dataset and a query workload. WISK is built in two stages. First, a partitioning algorithm finds the data clusters that minimize the time cost of executing the query workload. Then, an RL-based algorithm packs the data clusters into a hierarchical index in a bottom-up manner for more efficient pruning at query time. Learning from the query workload enables WISK to significantly outperform traditional SKR indexes. Experimental results on real-world datasets show that WISK yields strong query performance over various workloads, achieving up to  $8\times$  times speedups with comparable storage overhead.

There are several directions for future work. First, we intend to better answer Boolean  $k$ NN queries and to support more types of spatial keyword queries. Second, even WISK can adapt to workload changes by model retraining, a data-driven learned index, which is compatible with frequent workload shifts and data changes, is an interesting direction to explore.

## ACKNOWLEDGMENTS

This work was partially supported by Australian Research Council (ARC) Discovery Project DP230101534, Australian Research Council Future Fellowship FT210100303. This work was also supported in part by NSFC under Grant 62102341, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2022A1515010166, Shenzhen Science and Technology Program under Grants JCYJ20220530143602006 and ZDSYS2021102111415025, and Guangdong Talent Program under Grant 2021QN02X826. The authors would like to thank the anonymous reviewers for providing constructive feedback and valuable suggestions.

## REFERENCES

- [1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. 207–216.
- [2] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, A Inkeri Verkamo, et al. 1996. Fast discovery of association rules. Vol. 12. AAAI/MIT Press Menlo Park, CA, 307–328.



- [3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R\*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*. 322–331.
- [4] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2021. Machine learning for combinatorial optimization: a methodological tour d’horizon. *European Journal of Operational Research* 290, 2 (2021), 405–421.
- [5] Zdravko Botev and Ad Ridder. 2017. Variance reduction. *Wiley statsRef: Statistics reference online* (2017), 1–6.
- [6] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2001. STHoles: A multidimensional workload-aware histogram. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. 211–222.
- [7] Xin Cao, Gao Cong, Christian S Jensen, and Beng Chin Ooi. 2011. Collective spatial keyword querying. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. 373–384.
- [8] Yankai Cao and Victor M Zavala. 2020. A sigmoidal approximation for chance-constrained nonlinear programs. *arXiv preprint arXiv:2004.02402* (2020).
- [9] Huanhuan Chen, Peter Tiño, and Xin Yao. 2013. Efficient probabilistic classification vector machine with incremental basis function selection. *IEEE Transactions on Neural Networks and Learning Systems* 25, 2 (2013), 356–369.
- [10] Lisi Chen, Gao Cong, Christian S Jensen, and Dingming Wu. 2013. Spatial keyword query processing: An experimental evaluation. *Proceedings of the VLDB Endowment* 6, 3 (2013), 217–228.
- [11] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. 2006. Efficient query processing in geographic web search engines. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. 277–288.
- [12] Zhida Chen, Lisi Chen, Gao Cong, and Christian S Jensen. 2021. Location-and keyword-based querying of geo-textual data: a survey. *The VLDB Journal* 30, 4 (2021), 603–640.
- [13] Maria Christoforaki, Jinru He, Constantinos Dimopoulos, Alexander Markowetz, and Torsten Suel. 2011. Text vs. space: efficient geo-search query processing. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. 423–432.
- [14] Gao Cong, Kaiyu Feng, and Kaiqi Zhao. 2016. Querying and mining geo-textual data for exploration: Challenges and opportunities. In *2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 165–168.
- [15] Gao Cong, Christian S Jensen, and Dingming Wu. 2009. Efficient retrieval of the top-k most relevant spatial web objects. *Proceedings of the VLDB Endowment* 2, 1 (2009), 337–348.
- [16] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishé. 2008. Keyword search on spatial databases. In *2008 IEEE 24th International Conference on Data Engineering (ICDE)*. IEEE, 656–665.
- [17] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. 2020. ALEX: an updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 969–984.
- [18] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: a learned multi-dimensional index for correlated data and skewed workloads. *Proceedings of the VLDB Endowment* 14, 2 (2020), 74–86.
- [19] Xiaofeng Ding, Yinting Zheng, Zuan Wang, Kim-Kwang Raymond Choo, and Hai Jin. 2022. A learned spatial textual index for efficient keyword queries. *Journal of Intelligent Information Systems* (2022), 1–25.
- [20] Yixiang Fang, Reynold Cheng, Gao Cong, Nikos Mamoulis, and Yun Li. 2018. On spatial pattern matching. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 293–304.
- [21] Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1162–1175.
- [22] Saheli Ghosh, Tin Vu, Mehrad Amin Eskandari, and Ahmed Eldawy. 2019. UCR-STAR: The UCR spatio-temporal active repository. *SIGSPATIAL Special* 11, 2 (2019), 34–40.
- [23] Tu Gu, Kaiyu Feng, Gao Cong, Cheng Long, Zheng Wang, and Sheng Wang. 2023. The RLR-Tree: A Reinforcement Learning Based R-Tree for Spatial Data. *Proceedings of the ACM on Management of Data* 1, 1 (2023).
- [24] Jun Han and Claudio Moraga. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on Artificial neural networks*. Springer, 195–201.
- [25] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM SIGMOD Record* 29, 2, 1–12.
- [26] Ramaswamy Hariharan, Bijit Hore, Chen Li, and Sharad Mehrotra. 2007. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*. IEEE, 16–16.
- [27] Tuan-Anh Hoang-Vu, Huy T Vo, and Juliana Freire. 2016. A unified index for spatio-temporal keyword queries. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. 135–144.
- [28] Xiao Hu, Yuxi Liu, Haibo Xiu, Pankaj K. Agarwal, Debmalaya Panigrahi, Sudeepa Roy, and Jun Yang. 2022. Selectivity Functions of Range Queries are Learnable. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, Zachary Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 959–972.

- [29] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.
- [30] Xiaomian Kang, Yang Zhao, Jiajun Zhang, and Chengqing Zong. 2020. Dynamic Context Selection for Document-level Neural Machine Translation via Reinforcement Learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2242–2254.
- [31] Ali Khodaei, Cyrus Shahabi, and Chen Li. 2010. Hybrid indexing and seamless ranking of spatial and textual features of web documents. In *International Conference on Database and Expert Systems Applications*. Springer, 450–466.
- [32] Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020. RadixSpline: a single-pass learned index. In *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–5.
- [33] Stephen Cole Kleene. 1952. Introduction to metamathematics. (1952).
- [34] Taisuke Kobayashi and Wendyam Eric Lionel Ilboudo. 2021. T-soft update of target network for deep reinforcement learning. *Neural Networks* 136 (2021), 63–71.
- [35] Samuel Kotz, Tomasz Kozubowski, and Krzysztof Podgórski. 2001. *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Springer Science & Business Media.
- [36] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*. 489–504.
- [37] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. AI meets database: AI4DB and DB4AI. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*. 2859–2866.
- [38] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. 2020. LISA: A learned index structure for spatial data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2119–2133.
- [39] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
- [40] 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.
- [41] Ahmed R. Mahmood and Walid G. Aref. 2019. *Scalable Processing of Spatial-Keyword Queries*. Morgan & Claypool Publishers.
- [42] Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. 2020. Benchmarking learned indexes. *Proceedings of the VLDB Endowment* 14, 1 (2020), 1–13.
- [43] Francisco S Melo. 2001. Convergence of Q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep* (2001), 1–4.
- [44] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fiedjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
- [45] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning multi-dimensional indexes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 985–1000.
- [46] Andrew Ng, Michael Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems* 14 (2001).
- [47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32.
- [48] Kostas Patroumpas, Dimitrios Skoutas, Georgios Mandilaras, Giorgos Giannopoulos, and Spiros Athanasiou. 2019. Exposing points of interest as linked geospatial data. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases*. 21–30.
- [49] Marcelo Prates, Pedro HC Avelar, Henrique Lemos, Luis C Lamb, and Moshe Y Vardi. 2019. Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4731–4738.
- [50] Martin L Puterman. 2014. Markov decision processes: discrete stochastic dynamic programming. (2014).
- [51] Jianzhong Qi, Guanli Liu, Christian S Jensen, and Lars Kulik. 2020. Effectively learning spatial indices. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2341–2354.
- [52] Joao B Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørnvåg. 2011. Efficient processing of top-k spatial keyword queries. In *International Symposium on Spatial and Temporal Databases*. Springer, 205–222.
- [53] David Saad. 1998. Online algorithms and stochastic approximations. Vol. 5. Cambridge Univ. Press Cambridge, UK, 6.
- [54] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

- [55] Gleb Skobeltsyn, Toan Luu, Ivana Podnar Zarko, Martin Rajman, and Karl Aberer. 2007. Web text retrieval with a p2p query-driven index. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and Development in Information Retrieval*. 679–686.
- [56] Liwen Sun, Michael J Franklin, Sanjay Krishnan, and Reynold S Xin. 2014. Fine-grained partitioning for aggressive data skipping. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. 1115–1126.
- [57] Richard S Sutton and Andrew G Barto. 1999. Reinforcement learning: An introduction. *Robotica* 17, 2 (1999), 229–235.
- [58] Panagiotis Tampakis, Dimitris Spyrellis, Christos Doulkeridis, Nikos Pelekis, Christos Kalyvas, and Akrivi Vlachou. 2021. A Novel Indexing Method for Spatial-Keyword Range Queries. In *17th International Symposium on Spatial and Temporal Databases*. 54–63.
- [59] Hannu Toivonen. 2010. Frequent Itemset. In *Encyclopedia of Machine Learning*, Claude Sammut and Geoffrey I. Webb (Eds.). Springer, 418.
- [60] William TB Uther and Manuela M Veloso. 1998. Tree based discretization for continuous state space reinforcement learning. *Aaai/iaai* 98 (1998), 769–774.
- [61] Subodh Vaid, Christopher B Jones, Hideo Joho, and Mark Sanderson. 2005. Spatio-textual indexing for geographical search on the web. In *International Symposium on Spatial and Temporal Databases*. Springer, 218–235.
- [62] Haixin Wang, Xiaoyi Fu, Jianliang Xu, and Hua Lu. 2019. Learned index for spatial queries. In *2019 20th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 569–574.
- [63] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *Proc. VLDB Endow.* 14, 9 (2021), 1640–1654.
- [64] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. 2014. Selectivity estimation on streaming spatio-textual data using local correlations. *Proceedings of the VLDB Endowment* 8, 2 (2014), 101–112.
- [65] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. 2014. Selectivity estimation on streaming spatio-textual data using local correlations. *Proceedings of the VLDB Endowment* 8, 2 (2014), 101–112.
- [66] Zheng Wang, Cheng Long, and Gao Cong. 2021. Trajectory simplification with reinforcement learning. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 684–695.
- [67] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (1992), 279–292.
- [68] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S Jensen. 2011. Joint top-k spatial keyword query processing. *IEEE Transactions on Knowledge and Data Engineering* 24, 10 (2011), 1889–1903.
- [69] Jiacheng Wu, Yong Zhang, Shimin Chen, Jin Wang, Yu Chen, and Chunxiao Xing. 2021. Updatable learned index with precise positions. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1276–1288.
- [70] Wentao Wu, Chi Wang, Tarique Siddiqui, Junxiong Wang, Vivek Narasayya, Surajit Chaudhuri, and Philip A Bernstein. 2022. Budget-aware Index Tuning with Reinforcement Learning. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*. 1528–1541.
- [71] Dingqi Yang, Bingqing Qu, Jie Yang, and Philippe Cudre-Mauroux. 2019. Revisiting user mobility and social relationships in lbsns: a hypergraph embedding approach. In *Proceedings of the 30th International Conference on World Wide Web*. 2147–2157.
- [72] Yang Yang, Ying Zhang, Wenjie Zhang, and Zengfeng Huang. 2019. Gb-kmv: An augmented kmv sketch for approximate containment similarity search. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 458–469.
- [73] Zongheng Yang, Badrish Chandramouli, Chi Wang, Johannes Gehrke, Yinan Li, Umar Farooq Minhas, Per-Åke Larson, Donald Kossmann, and Rajeev Acharya. 2020. Qd-tree: Learning data layouts for big data analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 193–208.
- [74] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2013. Inverted linear quadtree: Efficient top k spatial keyword search. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 901–912.
- [75] Dongxiang Zhang, Kian-Lee Tan, and Anthony KH Tung. 2013. Scalable top-k spatial keyword search. In *Proceedings of the 16th International Conference on Extending Database Technology*. 359–370.
- [76] Lixi Zhang, Chengliang Chai, Xuanhe Zhou, and Guoliang Li. 2022. LearnedSQLGen: Constraint-aware SQL Generation using Reinforcement Learning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 945–958.
- [77] Songnian Zhang, Suprio Ray, Rongxing Lu, and Yandong Zheng. 2021. SPRIG: A Learned Spatial Index for Range and kNN Queries. In *17th International Symposium on Spatial and Temporal Databases*. 96–105.
- [78] Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. 2021. Online 3D bin packing with constrained deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 741–749.
- [79] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. 2005. Hybrid index structures for location-based web search. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*. 155–162.

Received October 2022; revised January 2023; accepted February 2023