Learned Indexes: A Survey of Methods and Comparisons

Haoxiang Wang, Chao Zhou

Department of Computer Science, Syracuse University

December 2022

Abstract

With the big boom of data and artificial intelligence technologies, more efficient index structures still need exploration. One of the solutions is learned indexes. Learned indexes are models that utilize deep learning or machine learning techniques to understand the data patterns and predict records' positions effectively. Although learned indexes have become popular in recent years, issues exist in reproduction, implementation, and datasets. One of our contributions is surveying the different supervised and reinforcement learning learned index methods. Moreover, we discuss the similarity and differences between both two mainstream direction and their potential issues and improvements.

Keywords— Leaned indexes, Survey, Methods, Comparisons

1 Introduction and Background

A learned index is a type of index structure in a database management system (DBMS) created and maintained by a machine learning algorithm. In contrast to traditional indexes, which are created and maintained manually by a database administrator, a learned index is automatically generated based on the data stored in the database and the specific access patterns of the queries being executed against the database. The goal of a learned index is to improve the performance of database queries by providing a more efficient indexing structure than what is possible with a traditional index. This is achieved by using machine learning algorithms to analyze the data in the database and the access patterns of the queries and then using this information to create an index structure optimized for the database's specific workload. Overall, a learned index can provide significant performance benefits for a database management system, particularly when dealing with large and complex datasets. It can also help to reduce the workload of database administrators, as the index is generated and maintained automatically by the machine learning algorithm. In general, machine learning algorithms are usually divided into supervised, unsupervised, and reinforcement learning methods. Supervised learning and reinforcement learning are widely used in the research on learned indexes because they have an advantage over unsupervised learning methods in capturing the semantic feature of data patterns.

Supervised learning is a type of machine learning in which a model is trained to make predictions based on labeled input data. In the context of a learned index, supervised learning would be used to train a model to create an index structure that is optimized for a specific workload of a database management system. The earliest learned index model was proposed by **Kraska et al. 2018**. Since then, scholars have proposed a variety of models for learned indexes, such as B+Tree-like learned index (**Ding et al. 2020**), single-layer adaptive learned index (**X. Li, J. Li, and X. Wang 2019**), learned index with adjustable error (**Galakatos et al. 2019**). To create a supervised learned index, the first step would be to collect a large amount of labeled data that is representative of the database's workload. This data would include the data stored in the database and information about the access patterns of the queries being executed against the database. Next, a machine learning algorithm would be trained on this data using supervised learning techniques. This would involve providing the algorithm with examples of input data and the corresponding correct outputs and then using this training data to fine-tune the model's parameters. Once the model has been trained, it can be used to generate an index structure for the database that is optimized for the specific workload of the database. The database management system can then use this index structure to improve the performance of queries being executed against the database. This can provide significant performance benefits for the database and can help to reduce the workload of database administrators.

On the other hand, reinforcement learning uses an agent to learn to make decisions in an environment by receiving rewards or punishments for its actions. Researchers applied reinforcement learning to train an agent to create an index structure optimized for a specific workload of a database management system. The mainstream reinforcement learning used in the learned indexes area is index selection (**Sadri, Gruenwald, and Leal 2020; Lan, Bao, and Peng 2020**), cost estimation (**Wu et al. 2022**), and joint query prediction and optimization (**Krishnan et al. 2018**). Despite the differences in their respective goals, the general process of training reinforcement learning is similar. To create a reinforcement learning learned index, the first step would be to define the environment in which the agent will operate. In this case, the environment would be the database management system, and the actions available to the agent would be the various ways in which it could create and maintain an index structure. Next, the agent would be trained using reinforcement learning techniques. This would involve providing the agent with a set of initial actions to take and then allowing it to interact with the environment and receive rewards or punishments based on its actions. Over time, the agent would learn which actions are most likely to lead to positive rewards and use this knowledge to make better decisions about creating and maintaining the index structure. Once the agent has been trained, it can be used to generate an index structure for the database that is optimized for the specific workload of the database. The database management system can use this index structure to make queries run more efficiently.

The main contributions of our survey lie in answering some significant questions in this area:

1. What is the supervised learning learned index? What is the mainstream of the supervised learning learned index? What is the similarity and difference between them?
2. What is the reinforcement learning learned index? What is the mainstream of the supervised learning learned index? What is the similarity and difference between them?
3. Why do we choose supervised learning or reinforcement learning learned index? What are their advantages and disadvantages of them? What is the similarity and difference between them?
4. What can we do to improve both methods? What can we learn from both methods?

To answer the above questions, we organize our survey as following structure: Section 2 mainly covers supervised learning methods and their comparisons. Section 3 mainly covers reinforcement learning methods and their comparisons. Section 4 discusses four areas of both two methods: dataset and workload, implementation, training cost, and evaluation methods. Section 5 summarizes our survey and propose some potential improvement direction in future.

2 Supervised Learning Methods and Comparisons

In recent years, researchers used supervised machine learning methods to optimize traditional data structures and designed a new class of index structures called the learned index. The learned index uses machine learning to identify patterns and trends in input data to predict the location of records. The learned index usually has smaller time complexity and space complexity compared with the traditional index. All learned indexes leverage a key design idea of using machine learning models to replace classic data structure building blocks such as B+tree. These models can adopt a more flexible approach to the data pattern.

2.1 Supervised Learning Methods on Learned Indexes

The most common structure of learned indexes is the tree. **Kraska et al. 2018** first proposed the learned index and opened a new research direction for building indexes. Their main contribution is to propose the Recursive Model Index (RMI), as shown in Figure 1. It is a hierarchy of models like a tree. Whenever a query is made, start from the model of the root node, and select the model adopted by the next level according to the input until the last layer model predicts the range of the position. The construction of RMI proceeds top-down through the hierarchy of training regression models. Firstly, use the whole data set D to train Model 1.1 and divide D into several subsets $D_1, D_2, \ldots, D_i$. Secondly, train the model Model 2.1, Model 2.2, ..., Model 2.i in the second level. This process is performed recursively until the last layer, and the data is continuously subdivided in this process.

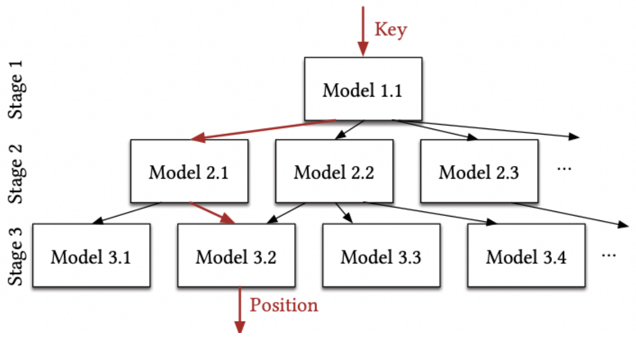

Figure 1: Recursive Model Index

However, a critical drawback of **Kraska et al. 2018** is that it lacks support for insertion and deletion, which makes it unusable in the dynamic read-write workflow. To solve this defect, **Ding et al. 2020** proposed the Adaptive LEarned indeX model ALEX. ALEX uses a B+tree-like structure to optimize the RMI model to support update operations. ALEX has two kinds of nodes, internal nodes and data nodes. Internal nodes are nodes in the RMI structure, as shown in Figure 2. They store a linear regression model and an array containing pointers to children nodes. For the key space in each internal node, divide it equally into power-of-two key spaces. For each key space, if the cumulative distribution function (CDF) is linear, assign it to a data node; if the CDF in it is non-linear and requires more resolution, assign it to another internal node. Data nodes are the leaf nodes in the model. It stores a linear regression model (two double values for slope and intercept), which maps a key to a position and two Gapped Arrays (keys and workloads). Gapped Arrays are used to absorb inserts like "free space" at the end of the array in each leaf node in B+Tree. When inserting an element, if the corresponding gap is not full, ALEX inserts it directly into this gap; if the corresponding gap is full, ALEX uses expansions and splits to create more space. With this structure, ALEX can achieve faster insert and lookup operations.

X. Li, J. Li, and X. Wang 2019 proposed a novel, simple learned index called Adaptive Single Layer Model (ASLM) in 2019, as shown in Figure 3. It is a single model with K sub-models in one layer. The training process of this model can be divided into three steps:

1. Specify the number of subsets (the value of K).
2. Data partition. Sort the data by the value of the key. Consider the key and position of the data and calculate the Euclidean distance (or the area of the triangle) between adjacent data. Select the largest K-1 distance (areas) as the splitting points and divide the workload into K pieces.
3. Single layer model training. Distribute the K pieces of data to K sub-models. Use the same network structure for all the sub-models. Train the K sub-models and initialize the next sub-model with the previous weights.

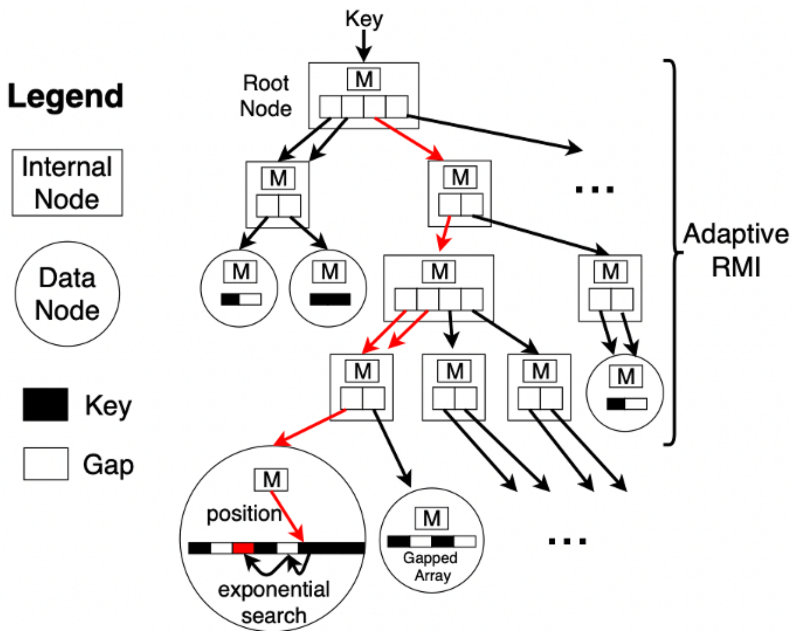

Figure 2: ALEX Model

To support model updating, each model needs to maintain a buffer that stores updated data. The buffer is organized like a hash table. When it is full, use the data in the buffer to retrain the sub-model.

Another learned index model FITing-Tree with an adjustable error parameter was proposed by **Galakatos et al. 2019**. The core of the FITing-Tree is the tunable error parameter (the maximum error allowed between the predicted and actual positions). For a given workload, FITing-Tree partitions its key space into variable-sized segments that satisfy the specified error, then inserts the data into the B+ tree in units of segments for efficient insertion and lookup operations. Besides, **Galakatos et al. 2019** proposed a cost model to help the FITing-tree determine the best error parameter for the given workload in the real world. This cost model can calculate the error parameter based on the allowable lookup latency or index size. However, they did not give the method to delete or update an index in the Fitting-Tree.

2.2 Supervised Learning Methods Comparisons

The Learned Index proposed by **Kraska et al. 2018** is a heavy multilevel model spending a lot of time on building and training. Another critical drawback is that RMI lacks support for insertion and deletion, which makes it unusable in the dynamic read-write workflow.

The ALEX model combines the advantages of B+Tree and the Learned Index proposed by **Kraska et al. 2018**. In terms of storage layout, ALEX adopts the B+Tree-like layout. ALEX implements fast insertion and deletion operations by growing, shrinking, and splitting nodes. In the search method, ALEX uses the linear regression model to predict the data position. If the model is accurate, it can achieve higher search efficiency and a smaller index size than B+Tree. Empirical experiments show that on various workloads, compared with B+Tree, ALEX has 4.1× higher search efficiency and 800× smaller index size on read-only workloads, and has 4.0× higher search efficiency and 2000× smaller index size on read-write workloads. Compared with the Learned Index, ALEX has 2.2× higher search efficiency and 15× smaller index size on read-only workloads.

Compared with the work of **Kraska et al. 2018**. The ASLM model by **X. Li, J. Li, and X. Wang**

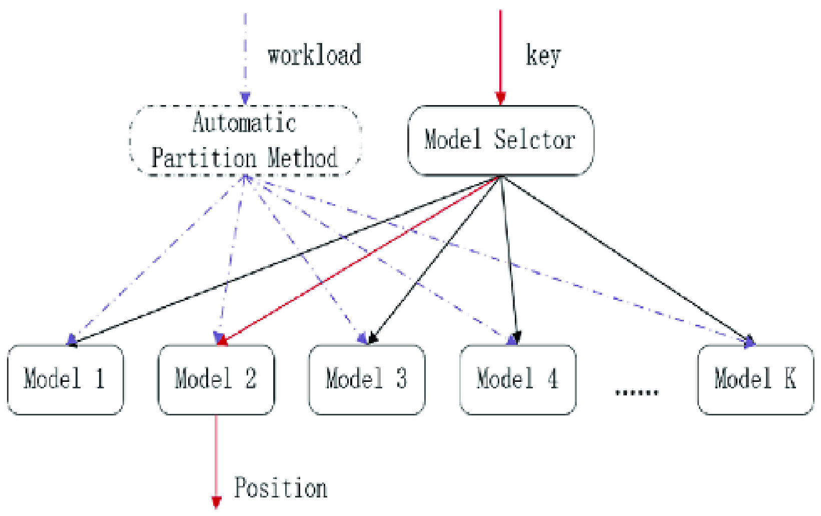

Figure 3: Adaptive Single Layer Model

2019 has three advantages. First, ASLM spends less time on training and querying because ASLM is a single-level model, and tree structure learned index has at least two layers. Second, ASLM uses the correlation between data and has a better data partition strategy to reduce the prediction error. Third, ASLM supports fast update operations since each sub-model has a buffer and does not need to be retrained immediately. When the buffer is full, ASLM only needs to retrain one or two sub-models. Compared with the ALEX model by **Ding et al. 2020**, ASLM's training cost advantage remains. However, The fitting ability of single-layer learned index is not as good as that of multi-layer, and it may be difficult to predict accurately when the data pattern is complex. In addition, the setting of the K value will also have a great impact on the accuracy of the model. Setting k too high will lead to a waste of time in the boundary checking process. Setting k too small can lead to inaccurate models because the workload is not well distributed. Which model to choose should be based on the specific conditions of the workload.

3 Reinforcement Learning Methods and Comparisons

Recent research aims to improve the performance of a learned index in real databases by using reinforcement learning (RL) paradigms. Although detailed implementations targeting using reinforcement learning are diverse among research, the backbone structure and core idea are similar to each other in that training a search policy to find a near-optimal combination plan over the existing index structures. One of the common assumptions that all these implementations should base on is that the users can provide a data pattern or a function to sort or partition the data, which is very much valid for most workloads. Therefore, most research doing this work tended to apply Recurrent Neural Networks (RNNs) or their variant version LSTMs (Long Short-Term Memory) as backbones. It mainly benefited from three advantages: 1. RNN can help researchers construct a tree-like index layer-by-layer way, where each layer is a sequence of abstract index blocks partitioning the search space with a predefined function. 2. RNN can search the optimal configuration for each index block, including block type, block size, the minimal and maximal number of keys in a block, etc. 3. RNN performs better in dealing with the long sequence and is more efficient in extracting long-dependence relationships within the sequence, which helps establish a general data pattern. Our survey mainly reviews four reinforcement learning-based learned index methods to show different optimization strategies.

3.1 Reinforcement Learning Methods on Learned indexes

Lan, Bao, and Peng 2020 proposed a reinforcement learning approach to find a proper index. In contrast to existing machine-learning approaches, they integrate index recommendation rules and deep reinforcement learning. Their paper proved that reinforcement learning has an advantage in foreseeing the global impact of introducing new indexes over traditional greedy-based methods. In the implementation section, they design five heuristic rules to generate the index candidates and a reinforcement learning model to select a subset of candidates as the recommended indexes. With the help of the Deep Q Network (DQN, Mnih et al. 2015), they modeled the interactions between indexes as a feature representation of index structure. The feature representation was applied to organize the target workload's indexes structure. The system runs the target workload under the indexes structure and returns the cost to the RL framework to improve the representations.

Similar to **Lan, Bao, and Peng 2020**, **Sadri, Gruenwald, and Leal 2020** also adopted deep reinforcement learning as the backbone of their index selection method for a cluster database. However, the heuristic rules are not common sense among researchers. **Sadri, Gruenwald, and Leal 2020** pointed out that These heuristic approaches do not have a mechanism to learn about the goodness of the recommended index set. Thus, they might choose the same index set with a low impact on I/O cost reduction. They applied reinforcement learning to a distributed database. Figure 4 shows the procedure of Index selection for a distributed database. Although they were applied to different systems, the idea between these two approaches is the same using reinforcement learning to capture the query pattern and apply them to systems to get rewards for improvement.

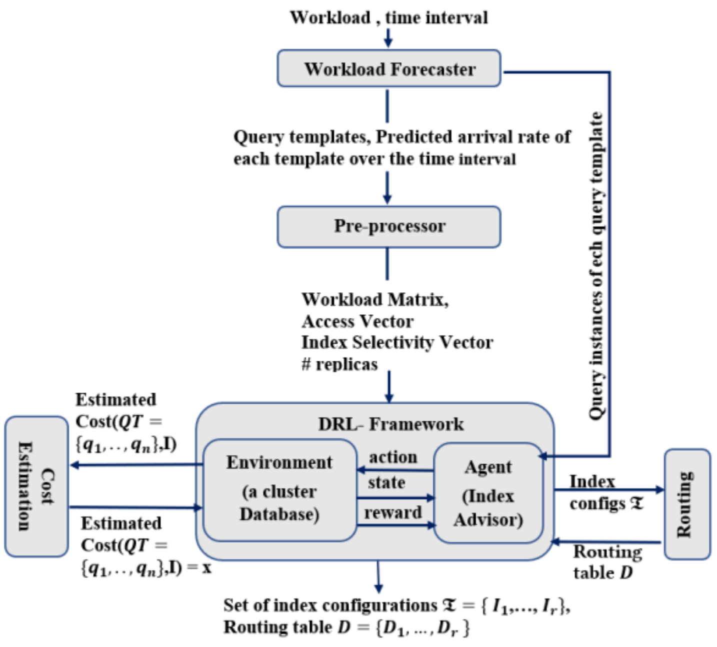
Workload , time interval
Workload Forecaster
Query templates, Predicted arrival rate of each template over the time interval
Pre-processor
Workload Matrix, Access Vector Index Selectivity Vector # replicas
Query instances of ech query template
DRL- Framework
Environment (a cluster Database)
action
state
reward
Agent (Index Advisor)
Index configs $\mathfrak{I}$
Routing
Routing table D
Cost Estimation
Estimated Cost(QT = {q_1,..,q_n},I)
Estimated Cost(QT = {q_1,...,q_n},I) = x
Set of index configurations $\mathfrak{I} = \{I_1,\ldots,I_r\}$, Routing table $D = \{D_1,\ldots,D_r\}$

Figure 4: The DRL-Index Selection Procedure for a distributed database.

Differing from **Lan, Bao, and Peng 2020** and **Sadri, Gruenwald, and Leal 2020**, **Wu et al. 2022** followed the Learned Index Structures developed by **Kraska et al. 2018** and provided a new structure named NIS. Unlike previous work that suffered a lot from the slow prediction of neural networks and scarce

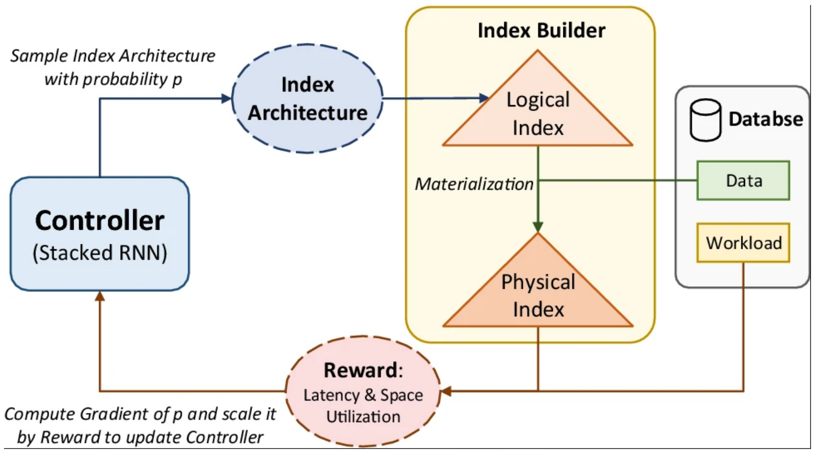
Sample Index Architecture with probability p
Index Architecture
Index Builder
Logical Index
Materialization
Physical Index
Databse
Data
Workload
Controller
(Stacked RNN)
Reward:
Latency & Space Utilization
Compute Gradient of p and scale it by Reward to update Controller

Figure 5: General architecture of NIS.

training recourse, NIS adopts a different strategy by searching for a solution to combine existing index blocks and tune their parameters for specific applications. One of the contributions of this work is reducing the prediction cost. In the implementation part, NIS chose stacked LSTMs as the backbone and tuned the database using reinforcement learning. Figure 5 shows the general architecture. NIS took the reinforcement-learning model as a controller to generate the architectural parameters of an index. After generation, NIS executes a specific query workload W on the database applying the architectural parameters and getting latency and space utilization as rewards. Then, NIS updates the policy according to the rewards until the policy converges. Finally, NIS will return a result of the target search policy with the corresponding workload.

Another way to improve the learned index structure is by exploring the query pattern in reinforcement learning. Join queries are much slower than other query types because of their combinatorial complexity, and it was usually avoided. The work of **Krishnan et al. 2018** proposed a new query optimizer that synthesizes datasets-specific join search strategies using RL. The core idea is learning a search strategy from the outcomes of previous planning instances that dramatically reduce search time for future planning. One of the benefits of this approach is that anywhere an enumeration algorithm is used, a policy learned from an RL algorithm can just as easily be applied. They chose a multi-layer perceptron (MLP) neural network in the implementation part. They followed the Q-learning method, which can learn from offline data consisting of a hierarchy of optimal sub-plans. It is better to fit for join optimization. One of their contributions is that they provide a method to feature the join decision and convert it into a vector for training models.

3.2 Reinforcement Learning Methods Comparisons

Empirical experiments show the shared performance that deep reinforcement learning has an advantage over machine learning approaches in capturing the patterns of query and index structure. The underly idea of these research is adopting reinforcement learning as a feature representation transformation tool. Another notable point is the training datasets used in most reinforcement learning-related research. **Wu et al. 2022** employ four datasets for evaluation and training: a large synthesis uniform dataset (*uniform64*) and three real datasets (*amzn, facebook, osmc*) **Kipf et al. 2019**.**Lan, Bao, and Peng 2020** employs a 1GB TPC-H database with eight tables and two synthetic workloads. **Krishnan et al. 2018** uses a Join Order Benchmark (JOB), which is 3.6G and 12 tables. All these datasets are applied directly to databases and perform well on different workloads. Despite the benefits and advantages RL brings, these methods are all time-consuming, which should be solved in the future. Their methods also contain limits on some aspects. For example, **Lan, Bao, and Peng 2020** combined heuristic rules and deep learning methods, but their experiments didn't evaluate the improvements from heuristic rules. **Sadri, Gruenwald, and Leal 2020** only recommend single-attribute indexes for all kinds of queries which is not efficient for complex queries.

4 Discussion: Supervised Learning or Reinforcement Learning

4.1 Dataset and workload

The training datasets and testing datasets are significant for machine learning-related work. One of the limitations of the learned index is the scarce dataset. Thanks to the work of SOSD benchmark (**Kipf et al. 2019**), TPC benchmark, and other open-source datasets, the learned indexes methods could be made. Table 1 shows the datasets used in surveyed papers. Although the detailed research direction is various, the shared features of datasets are that all the training datasets have at least 200M records, and most of the testing datasets are based on real applications and databases. But the division still exists when we delve into the details about them. In the supervised learned index, a dataset is used to train a model to make predictions or classify data. The dataset is labeled, which means that the correct output or classification is provided for each example in the dataset. The model is trained on the labeled dataset and learns to predict the correct output for new examples based on the patterns identified in the training data. For example, in supervised learned indexes, a linear dataset is a dataset with uniform data distribution, such as longitude. While a non-linear dataset has complex data patterns that are difficult for machines to learn. The requirement for high-quality labeled datasets leads many supervised learned index methods to choose datasets from broader areas that are not satisfying the learned index task (Kipf et al. 2019). In reinforcement learning, a dataset is not used to train the model directly. Instead, the model learns to make decisions by interacting with its environment and receiving feedback in the form of rewards or punishments. This feedback is used to update the model's decision-making strategy in order to maximize the total reward over time. The model learns from its own experiences rather than from a labeled dataset. So real database datasets can be directly used in training models, which mitigates the issue. But we also see potential research using reinforcement learning to help with the automatically generated labeled datasets, which can be great for developing future supervised learned indexes. However, not all real database datasets can be applied to RL well. Most modern RL algorithms collect data episodically (execute an entire query plan and observe the final result). While episodical data ignore compositional structure, which is not suited for tasks consisting of sub-plans.

4.2 Implementation

Supervised learning learns the patterns and trends of data by pre-prepared labeled datasets and then predicts the location of the workload. All the supervised learning mentioned in this paper adopt the linear regression model as the structure of the sub-model. The difference is mainly in the hierarchy of nodes and the handling of update operations. **Kraska et al. 2018** proposed a simple tree structure that does not support update operations. **Ding et al. 2020** adopted a B+Tree-like structure and support update operations by growing, shrinking, and splitting nodes. **X. Li, J. Li, and X. Wang 2019** used a single-layer model to simplify the training process and update operations, at the cost of a decrease in the fit to irregular data. Figure 5 shows the general idea of reinforcement learning. One of the benefits brought by reinforcement learning is that reinforcement learning act as an agent independently and updates by rewards. **Wu et al. 2022; Lan, Bao, and Peng 2020; Sadri, Gruenwald, and Leal 2020** adopt stacked LSTM or deep Q net to guide the agents to solve the select indexes questions, while **Krishnan et al. 2018** uses a multi-layer perceptron neural network. Although deep neural networks increase the complexity of implementation, the interaction between agents and databases can be captured well. The modularization feature of RL methods has an advantage over supervised learning methods.

Table 1: Dataset used in learned indexes methods

Type	Model	Dataset Name	Size	Distributed
Supervised	Kraska et al. 2018	website requests	200M	non-linear
Supervised	Kraska et al. 2018	longitude	200M	linear
Supervised	Kraska et al. 2018	Synthetic dataset	200M	non-linear
Supervised	Ding et al. 2020	longitudes	1B	linear
Supervised	Ding et al. 2020	longlat	200M	non-linear
Supervised	Ding et al. 2020	lognormal	200M	non-linear
Supervised	Ding et al. 2020	YCSB benchmark	200M	distributed
Supervised	X. Li, J. Li, and X. Wang 2019	OpenStreetMap	200M	linear
Supervised	X. Li, J. Li, and X. Wang 2019	TPC-H benchmark	14M	non-linear
Supervised	Galakatos et al. 2019	weblogs	715M	non-linear
Supervised	Galakatos et al. 2019	loT	5M	non-linear
Supervised	Galakatos et al. 2019	longitude	2B	linear
Reinforcement	Wu et al. 2022	uniform64	200M	not reported
Reinforcement	Wu et al. 2022	amzn	200M	not reported
Reinforcement	Wu et al. 2022	facebook	200M	not reported
Reinforcement	Wu et al. 2022	osmc	200M	not reported
Reinforcement	Lan, Bao, and Peng 2020	TPC-H benchmark	1G	not reported
Reinforcement	Krishnan et al. 2018	Join Order Benchmark	3.6G	not reported
Reinforcement	Krishnan et al. 2018	TPC-DS	not reported	not reported

4.3 Training cost

Assuming that the training time of a single model is linearly related to the size of the data set n, the training time of the RMI is $O(kn)$, where k is the number of layers. **Ding et al. 2020** proof that an ALEX can be constructed with depth no larger than $\lceil log_m p \rceil$ where m is the maximum node size and p is how many partitions the key space is divided into. Therefore, the training time of ALEX is $O(n\lceil log_m p \rceil)$. Assuming there are $\sqrt[3]{n}$ data nodes, the maximum number of layers of the tree is 2, which means that the training complexity is $O(2n)$. ASLM has the lowest complexity among these models. Thanks to the fact that ASLM is a single-layer model, ASLM has the complexity of $O(n)$. However, for reinforcement learning, training cost is still an open question. The question mainly lies in a few points. First, RL algorithms often require many trials or episodes to learn the index pattern effectively, which can take significant time to run. Second, RL algorithms can be computationally complex because of the complex relationship in the actual database, which can slow down the training process. Third, the reward from a database is not timely, numerous query operations should test it. Finally, fine-tuning the hyperparameters of an RL model can also be a time-consuming process, as it often requires running multiple experiments with different combinations of hyperparameters to find the best-performing configuration. How to improve the efficiency of the RL learned indexes should be studied in the future.

4.4 Evaluation Methods

Compared with traditional indexes, the advantages of learned indexes are lower query latency and smaller indexes. However, researchers should make a tradeoff between them. Lower query latency requires smaller prediction errors, while smaller indexes will cause larger prediction errors. Learned indexes should balance the query latency and index size and use them as evaluation indicators. Learned indexes methods usually apply real workloads to evaluate performance, in both two methods, they tested the query latency and prediction error under different index structures. In particular, supervised learning tested the index size. Besides, the training cost is not evaluated by both methods, which should also be considered in their evaluation.

5 Conclusion

Supervised and reinforcement learning learned index methods share many similarities, and both have advantages in different aspects. This survey mainly focuses on reviewing different methods and comparing their complexity. The lack of a labeled training dataset is the main issue for supervised learning methods, and time-consuming training cost is also a concern for RL methods. More deep-learning techniques can be brought into this research, such as adding prompting to enhance the pattern recognition ability during the training process. Besides, combining the techniques used in the two methods by comparing their similarities and differences is also a potential research direction.

References

Ding, Jialin et al. (2020). "ALEX: an updatable adaptive learned index". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 969–984.

Galakatos, Alex et al. (2019). "FITing-Tree: A Data-Aware Index Structure". In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Amsterdam, Netherlands: Association for Computing Machinery, pp. 1189–1206. ISBN: 9781450356435. DOI: `10.1145/3299869.3319860`. URL: `https://doi.org/10.1145/3299869.3319860`.

Kipf, Andreas et al. (2019). *SOSD: A Benchmark for Learned Indexes*. DOI: `10.48550/ARXIV.1911.13014`. URL: `https://arxiv.org/abs/1911.13014`.

Kraska, Tim et al. (2018). "The Case for Learned Index Structures". In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD '18. Houston, TX, USA: Association for Computing Machinery, pp. 489–504. ISBN: 9781450347037. DOI: `10.1145/3183713.3196909`. URL: `https://doi.org/10.1145/3183713.3196909`.

Krishnan, Sanjay et al. (2018). "Learning to optimize join queries with deep reinforcement learning". In: *arXiv preprint arXiv:1808.03196*.

Lan, Hai, Zhifeng Bao, and Yuwei Peng (2020). "An Index Advisor Using Deep Reinforcement Learning". In: *Proceedings of the 29th ACM International Conference on Information &amp; Knowledge Management*. CIKM '20. Virtual Event, Ireland: Association for Computing Machinery, pp. 2105–2108. ISBN: 9781450368599. DOI: `10.1145/3340531.3412106`. URL: `https://doi.org/10.1145/3340531.3412106`.

Li, Xin, Jingdong Li, and Xiaoling Wang (2019). "ASLM: Adaptive single layer model for learned index". In: *International Conference on Database Systems for Advanced Applications*. Springer, pp. 80–95.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Sadri, Zahra, Le Gruenwald, and Eleazar Leal (2020). "Online Index Selection Using Deep Reinforcement Learning for a Cluster Database". In: *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, pp. 158–161. DOI: `10.1109/ICDEW49219.2020.00035`.

Wu, Sai et al. (2022). "Dynamic Index Construction with Deep Reinforcement Learning". In: *Data Science and Engineering* 7.2, pp. 87–101.