IndexHorizon: A Deep Reinforcement Learning Solution for Index Selection

Arvindjit Singh
Data and Knowledge Engineering Dept
Otto Von Guericke Universitat
Magdeburg, Germany
arvindj1@st.ovgu.de

Balakrishnan Ramakrishnan
Digital Engineering Dept
Otto Von Guericke Universitat
Magdeburg, Germany
balakrishnan.ramakrishnan@st.ovgu.de

Bhuvanesh Leelakrishnan
Data and Knowledge Engineering Dept
Otto Von Guericke Universitat
Magdeburg, Germany
bhuvanesh.leelakrishnan@st.ovgu.de

Rajatha Nagaraja Rao
Data and Knowledge Engineering Dept
Otto Von Guericke Universitat
Magdeburg, Germany
rajatha1@st.ovgu.de

Rohan Das
Data and Knowledge Engineering Dept
Otto Von Guericke Universitat
Magdeburg, Germany
rohan.das@ovgu.de

Zeeshan Shaikh
Data and Knowledge Engineering Dept
Otto Von Guericke Universitat
Magdeburg, Germany
zeeshan.shaikh@st.ovgu.de

***Abstract*—Given a workload and a set of constraints, such as budget, it was the duty of the database administrator to find a set of physical structures that minimize the cost of query execution for a given database. This physical design process includes setting of database configurations such as indexes, buffer sizes and others that result in the minimization of query execution cost. In recent times, many attempts are being made to automate the physical design process for databases. This has led to a reduction in the involvement of the database administrator in selecting the physical structures for optimizing the workload performance, which can be termed as the NoDBA approach.**

This paper delivers a NoDBA approach using deep reinforcement learning to determine the physical database configuration. The highlights of this paper include, the adaptation of physical designers design strategies into our approach, integration of deep reinforcement learning agents designed for images into our task, performance comparison of the used agents with the state-of-the-art advisory tool and the ability of the agents to generalize on a given query set.

I. Introduction

It has been studied by Forbes [1] that 2.5 quintillion bytes of data are generated everyday. It is also found that data doubles every two years and the resulting deluge of data that we have witnessed in the recent years requires efficient storage and processing capacities from database management systems. Commercial DBMS are highly complex software systems and consists of many configuration parameters which needs to be adjusted, as per the user/workload/hardware requirements, to achieve maximum performance on the system. Usually, system administrators, mainly known as DBAs (Database Administrators), are responsible for monitoring different workloads and, taking help from state of the art tools to tweak the various configuration parameters of a DBMS in order to improve the performance for their respective systems.

[1]https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/2a435c460ba9

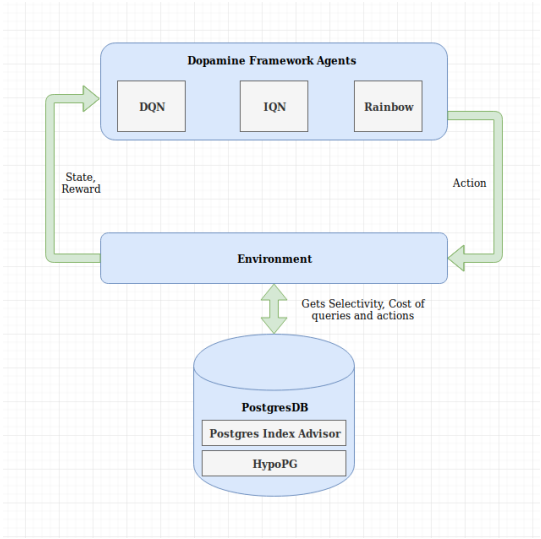

Fig. 1: Index Horizon Overview

In this paper we target one such configuration parameter, i.e. *Index Selection*. There are many ways to select indexes, such as manually (chosen by the DBA), partially automated (where the DBA uses supporting tools) and fully automated. In the manual selection, the DBA manually selects the indexes by analyzing different user requirements. In the partially automated approach, the DBA takes helps of various state of the art tools (i.e Index advisors) to select the optimal indexes. Finally, in fully automated index selection the indexes are selected automatically without the involvement of a DBA. Unlike the above two approaches, the automated method of index selection does not involve a DBA, it has been referred to as a NoDBA approach. Furthermore, fully automated selection of indexes has two more sub-approaches: Heuristic-driven [BAA12], and Application of machine learning algorithms. In

this paper, we focus on fully automated index selection, using machine learning approaches.

Machine Learning algorithms have an aim to learn a given task. Techniques such as reinforcement learning help the agents to learn the cost and rewards of various action selection strategies by learning in an ad-hoc manner rather than using pre-defined rules for data processing. Hence, the reinforcement learning agents can tune themselves to select the appropriate database design (such as index selection) based on the workload instead relying on the database administrator to do so. We focus on machine learning algorithms such as Deep Reinforcement Learning (DRL), to solve the task of automatic index selection which would give the maximum performance for different individual workloads. Unlike supervised learning, DRL does not need expected outputs in the training process, rather it uses a trial-and-error approach which involves rewards in the form of a feedback to the agent. The feedback is for the suggested action and can function as a positive or negative learning signal. We are using the above mentioned techniques with Google Dopamine [Cas+18] a state-of-the-art framework and compare the results generated by the agents of this framework against state-of-the-art index advisory tools.

In this paper we are trying to train different agents which are provided by the Dopamine framework, to perform automatic index selection. In this process, the database is used to give selectivity factors and cost of queries, whereas the index advisory is used as a baseline, to compare the indexes selected. As shown in Fig. 1, the agents in the framework interact with the environment by selecting an action to which the environment gives the reward and next state of the observation space. This process is repeated in a loop until the game end criteria is reached and the game finally terminates.

Though there has been previous research in applying DRL agents for index selection [SSD18], there is room for improvement in better understanding this approach. In our study, we explore this potential improvement area. Our core contributions are:

- We develop and evaluate a DRL solution for index selection, using standard DRL frameworks (and agent designs), in addition to a standard database benchmark, and an index advisory as a strong baseline. Furthermore, we take special care to replicate the experiment designs from state of the art evaluations of index advisory tools [BAA12].
- We evaluate whether the approach of using agent designs developed for images (e.g. the standard DQN, Rainbow and Implicit Quantile implementations), can contribute to learning index selection. We report on the training characteristics of these models, finding some variability on their convergence behaviour, which requires further study. Nonetheless, in all experiments, save for generalization, we found that an agent was able to converge successfully to the expected behavior.
- We compare the runtime of trained agents against that of index advisory tools, finding that the latter is notably slower in the task when compared to the pre-trained agents. We also validate that both approaches are able to provide the same recommendations.
- We conduct an early study concerning generalization of the agents, however we do not find a successful solution.
- We provide a comprehensive environment that can be adopted to evaluate further agents for the index selection task.

The rest of the paper is structured as, follows: In Sec. II, we introduce necessary background information, which includes a brief explanation on Index Advisory tools, and Deep Reinforcement learning. In Sec. III we present the experimental design and setup for our solution. Sec. IV includes the evaluation results for the experiments. Sec. V we have included the related work . Sec. VI includes the conclusion and the future work that needs to be done in the given field.

II. Background

A. Deep reinforcement learning

Deep reinforcement learning, which is a combination of reinforcement learning and deep learning, enables to perform complex tasks which involve decision making. The deep learning approach utilizes a deep neural network that contains successive layers of processing. In reinforcement learning, an agent learns good behaviour by trial-and-error experience. Given a state s_t, defined by its observation w_t, the agent interacts with the environment through a chosen action a_t, to receive a reward r_t. With this selected action the environment transitions to a new state $s_t + 1$, with its corresponding observation w_t+1, as shown in Fig. 2. The future state depends only on the current observation and the action, the full history is not considered, this design strategy is under the assumption that the environment follows the Markov Decision Process. In order to pick an action, the agent builds a state-action transition tree with each node containing the information on the expected rewards at that particular state for that particular action from that state. The way in which agents pick the actions, is called a policy. The policy used can either be deterministic or stochastic. Agents are expected to be rational, and they are designed such that they can learn a policy that maximizes the long-term cumulative reward.

Reinforcement learning can either take on a model-free (value-based or policy-gradient based) approach or a model-based approach. Value-based algorithms help to construct a value function, which eventually enables policy definition. Value-based learning algorithms include: Q learning, fitted Q learning and Deep Q Network(DQN). Policy-gradient based approaches focus on optimizing a performance objective such as the aggregate reward by finding explicitly what constitutes a good policy. They can work with continuous action spaces and are useful where the optimal policy is a stochastic policy, enabling exploration and working well with multi-agent scenarios. The model-based approach, in contrast to the model-free approach, requires a model of the transitions and rewards of the environment. This model can be explicitly defined or it can be learned through experience. After the model is learned,

a planning algorithm (controller) can use the model and derive the optimal actions all the time (the agent performs with the optimal policy).

In the subsequent sections, we will discuss how our environment is modelled along with the policies defined and the types of agents used. However in the next section we introduce the specific state-of-the-art index advisory tools.

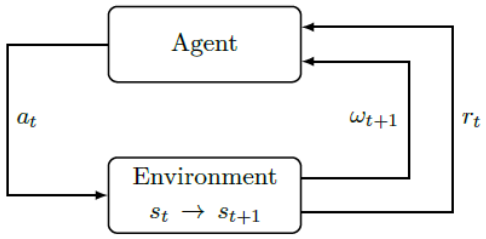

Fig. 2: Agent-Environment Interaction: The core abstraction for reinforcement learning

B. Index Advisory tools

Index Selection is one of the main parameters for a database system, as indexes contribute to improve the performance of queries. Apart from manual selection, index advisory tools can be used to recommend indexes. They provide a better way of selecting a good index for various configurations. The key function of these advisory tools is to recommend a set of indexes which can be used to optimize the performance, based on some expected group queries. An index may have multiple columns as key columns, and the ordering of those columns is significant. Finding the set of indexes that optimize a workload of complex, multi-table queries having varying importance and subject to resource constraints, is very challenging [Val+00]. For example, in a real world scenario the applications can contain thousands of tables and further each table can have hundreds of columns. For a workload there will be a large number of queries and to consider what could be the possible index combination (and their benefit) is not an easy task.

In the beginning these tools were totally different from the database engine and proposed independent set of indexes along with the cost evaluation. A major advance in the design tools was the use of the engines optimizer to evaluate the cost of queries, given a set of candidate indexes[FST88]. This was achieved through the use of hypothetical indexes.

Physical database design focuses on selecting the appropriate physical structures such as indexes and materialized views which aims to improve the database performance, with respect to a given workload [BAA12]. Chaudhuri et. al.[CN97] propose the AutoAdmin index selection tool in an attempt to automate the index selection process. AutoAdmin, implemented on the Microsoft SQL Server, removes spurious indexes using its candidate index selection module. The configuration enumeration module selects the index configuration with the lowest cost and the multi-column index generation module selects multi-column indexes. The ”what-if” index creation module is used to simulate the suggested indexes in the database. Hence, the paper proves that the indexes suggested by AutoAdmin improve the performance of the database drastically. Authors, such as Borovica et. al.[BAA12], have studied the accuracy of physical designers such as AutoAdmin, by checking if the estimated improvements suggested by it correspond to the actual improvements obtained by the database in real-time. Our confidence in the physical designers increase if they correctly estimate the impact of using the physical database designs suggested by them. After experimenting with TPC-H and NREF data sets, authors [BAA12] conclude that the estimates provided by the physical designers are inaccurate, inconsistent and are adversely affected by the cardinality errors of the query optimizer.

After introducing a basic background on deep reinforcement learning and index advisory tools, we can present our research questions in the following section.

III. Research Questions and Experimental Setup

In this section we define the research questions that guide our work, as follows:

1) Considering that index advisory tools have been shown to be unstable in their ability to find the optimal index configuration when either changing the threshold for the maximum number of possible indexes[BAA12], we propose to study: How does a learned tool with DRL compare against an index advisor under these changing settings?
2) How do different DRL agents compare with regards to their training requirements, for training on a workload and doing predictions on the same workload?
3) How do agents compare when they train on a workload and seek to generalize to other workloads?
4) How do DRL tools compare to an index advisor in the speed of the inference task?
5) How do DRL tools compare to the quality of indexes recommended by the advisory tools?

In order to answer to these research questions, we define an experimental setup, consisting of six major parts:

1) Queries
2) Database
3) Index Advisor
4) Dopamine Framework
5) Environment Design
6) Architecture

From these, the environment design corresponds to the design of our solution. In the following sections we describe these aspects.

A. Queries

We have tested our experiments with the TPC-H analytical benchmark as it is an accepted industry-standard benchmark for evaluating databases. TPC-H provides 22 standard queries, out of which we have chosen 7 queries to train and test

our agents[2]. Due to the complex nature of some of the TPC-H standard queries the index advisor used in the environment failed to suggest indexes. Even the recommended Postgresql parser, due to its recursive structuring degraded the performance of our environment while parsing these complex queries, which lead to exclusion of some complex queries. A custom parsing logic was developed for the 7 TPC-H queries under consideration to overcome the problem posed by the complex queries and the recommended parser. The 7 queries used have numbers as follows: 1, 3, 5, 6, 10, 12, 14.

B. Database

We have used the PostgreSQL database, which supports pluggable Index advisors, and is highly extensible, allowing us to define custom objects, functions and data types. Another reason to use Postgres was the availability of HypoPG for setting hyothetical indexes. The agents are tested in the Dopamine deep reinforcement learning framework. The size of our database for training is 1GB. During the set up we discovered that the TPC-H Scale Factor 1 generated different data in different systems due to which the number of indexes suggested by the index advisory differed from system to system. We tried to overcome this problem by replicating the same data among different systems but we faced the same issue of different indexes because the index advisory recommends indexes based on the configuration of the particular system.

C. IndexAdvisor

$Pg\_idx\_advisor$ is the index advisor tool that we have selected as a baseline for our experiments. $Pg\_idx\_advisor$ suggests indexes for different columns of the database tables. In our experiment we are retrieving simple indexes from the index advisor tool. We use these indexes as a baseline to compare the indexes provided by our agents.

D. Frameworks

The agents are tested in Dopamine, which is a state-of-the-art deep reinforcement learning framework. Dopamine, provided by Google, is an open source deep reinforcement learning framework based on TensorFlow. We test the DQN (Deep-Q-Network), Rainbow and Implicit Quantile agents of Dopamine. All these agents are variations of DQN. DQN agents are the preferred agents for Discrete action spaces.

E. Environment Design

The following custom environment was developed using the OpenAI Gym.

1) Observation Space: The observation space is a 2-dimensional matrix of shape (8,61). Each column in the matrix represents a column from a table in the database (these are the total number of columns in the TPC-H schema). The first row is a binary representation of the current state of the database indexes. To be more clear if there is an index set on a particular column in the database then it is represented by '1' for its respective column in the observation matrix. Similarly, the remaining rows contain the selectivity of the predicate columns of each query in its respective matrix columns. The input to generate the observation space is just the list of queries that are currently being used to train the agent.

$$W = \begin{bmatrix} i_{0,0} & i_{0,1} & i_{0,2} & \dots & i_{0,60} \\ sel(Q_{1,0}) & sel(Q_{1,1}) & sel(Q_{1,2}) & \dots & sel(Q_{1,60}) \\ sel(Q_{2,0}) & sel(Q_{2,1}) & sel(Q_{2,2}) & \dots & sel(Q_{2,60}) \\ \dots & \dots & \dots & \dots & \dots \\ sel(Q_{7,0}) & sel(Q_{7,1}) & sel(Q_{7,2}) & \dots & sel(Q_{7,60}) \end{bmatrix}$$

2) Action Space: The action space contains the actions that an agent can perform. In our case, the action space is discrete and contains 61 actions one for each column in the matrix. Each action is an instruction to the environment to set an index in the database, on the selected column.

3) Reward Function:

$$Reward = [\frac{Value_{agent} - Value_{initial}}{Value_{best} - Value_{initial}} X100] - No.of Steps$$

The reward function gives two forms of rewards one if the game end criteria is not reached and the other if it is reached. The latter is explained below.

Previous work [SSD18] has considered the cost of no indexes as a baseline in calculating the reward for some index selection process. This, however does not answer the question if the agent can replicate the method of indexing used by a human DBA. To answer this intuition instead of using a human feedback on which indexes to set based on the queries we have used an index advisor. The index advisor suggests indexes which is used to calculate the cost of running a set of queries. The reward consists of a value, which is nothing but the inverse of the cost values. The numerator consists of the difference between the value calculated based on the agents action and the initial value without any indexes. The denominator consists of the difference between the value returned by the index advisor indexes and the initial value. This fraction is then multiplied by 100 and subtracted by the number of steps it took to reach the game-end criteria.

We know that the goal of a reinforcement learning agent is to increase the reward. As we have a cumulative reward equation for the game end criteria, the only way to maximize the rewards is by increasing the numerator, which in turn increases the value of the agent thereby decreasing the cost of the agent. This means the lower the costs given by the agent the better the rewards. Also, this fraction is then subtracted by the number of steps taken to reach the game end criteria. This ensures that the goal state is reached in minimum number of steps.

4) Step Function: Each step involves forwarding the action given by the agent to the database, setting the respective index and returning a cost value (run time) of running the queries after setting the index. Using this cost we get the value. At this point the process can be divided into two parts:

- If a game end criteria is not reached then a reward of '1' is given back to the agent.

[2]http://www.tpc.org/tpch/

- If a game end criteria is reached then the reward is calculated based on the reward equation.

The game end criteria is attained if one of the below things occur:

- Same index is repeated.
- The number of indexes set goes beyond a certain threshold (k in our case).
- There is no improvement in the current cost compared to the previous action cost given by the agent.

5) Execution Flow: The very first step is to initialize the agent. The initialization is declaring the necessary variables.

The initial observation is given to the agent, which in turn returns an action. This action is given to the step function which gives the next state of the environment and the reward as a feedback.

If a game end flag was sent to the agent, then the environment resets to the original values, else the agent continues to send actions to the environment. Fig. 3 shows the overall architecture of our solution, describing how our environment integrates with the Dopamine framework and the role of the index advisor.

With this we conclude our experimental setup and design. In the following section we discuss our evaluation.

6) Architecture: The architecture in Fig. 3 explains how Dopamine framework integrates with our environment. Essentially, the framework has a central component in the form of a runner, which manages the complete training process. The runner acts as a middleware between agents and environment. There are also checkpointer and test logging components, which help to store the state of the training to disk.

We introduce an IndexSelection environment, which we will implement using OpenAI Gym. This environment connects to the database and to an index advisor tool, these two are used to calculate the rewards of actions mentioned in reward section III-E3 in Environment Design.

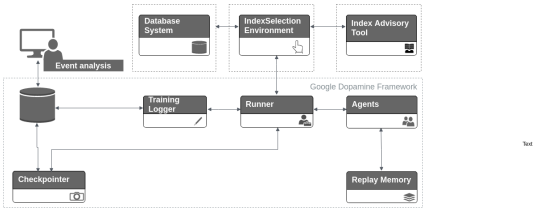

Fig. 3: Integration of Index Horizon with Dopamine

IV. Evaluation

We conducted the following different experiments on each agent for 500 iterations, which is a limited number of iterations when compared to the standards reported in DRL publications (i.e., millions of iterations), for the agents to learn, but due to system and time limitations we used only 500 iterations and the results for different experiments are as follows:

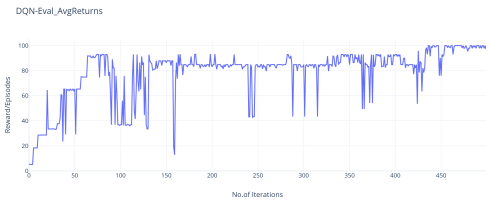

Fig. 4: DQN-EvalReturns k = 0

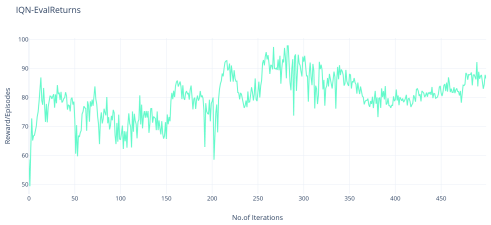

Fig. 5: IQN-EvalReturns k = 0

A. Experiment 1

In this experiment we are considering 7 TPC-H queries as mentioned in the Experimental Setup section. The queries are passed to the database and the agent is trained to index the database based on these 7 queries. In this experiment, we are over-fitting the agents and measuring the feasibility of learning i.e. whether the agents can learn the best indexes for these queries. We are also considering 3 different thresholds for the number of indexes where k is the number of indexes suggested by the index advisor for the 7 set of queries, we are running the agents with different values of k i.e. k, k+2, k-2 . With this experiment, we are trying to find out how different agents from different frameworks converge their learning with varying index values. The goals we try to achieve with this experiment are: Feasibility of learning process and comparison of different DQN-based agents on how their learning converges. This answers to our first and second research questions.

The agents train for a number of steps and then are evaluated for a number of steps. In our studies each episode consists of 15 steps, and we trained the agents on 1000 steps, and evaluated them on 550 steps, per iteration.

1) For k-offset=0: When running the queries for the agent DQN with k as the number of indexes we observe that DQN agents explore the possibilities of best indexes and with they

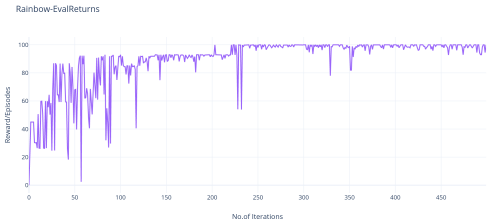

Fig. 6: Rainbow-EvalReturns k = 0

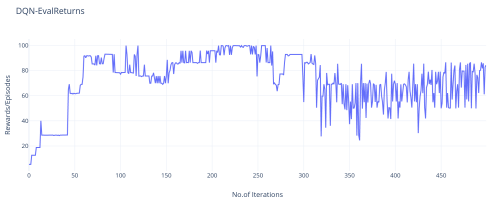

Fig. 7: DQN-EvalReturns k = +2

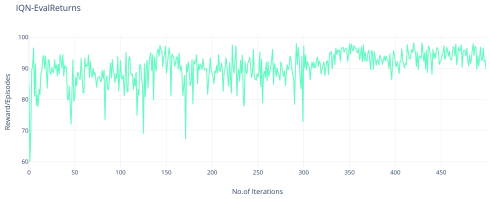

Fig. 8: IQN-EvalReturns k = +2

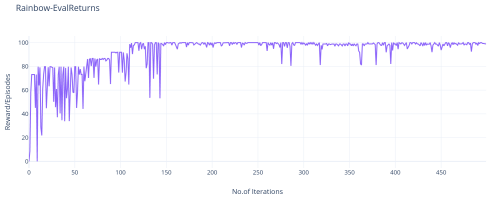

Fig. 9: Rainbow-EvalReturns k = +2

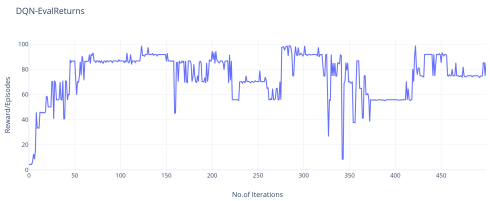

Fig. 10: DQN-EvalReturns k = -2

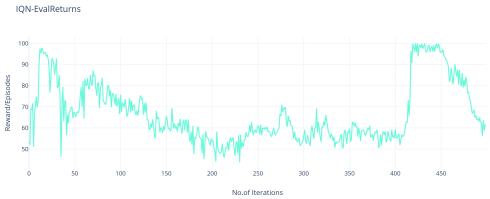

Fig. 11: IQN-EvalReturns k = -2

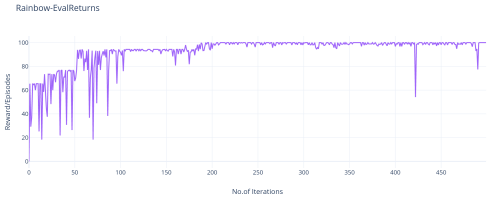

Fig. 12: Rainbow-EvalReturns k = -2

are eventually able to converge. As we can see in Fig. 4 as the number of iterations increase from 400 onwards, the agent starts converging and provides a reward of 100 and it tends to stay there, hence we can say that for our experiment DQN agent has found the optimal set of k indexes. Similarly, Fig. 6 shows that the Rainbow agents take even less number of iterations (i.e. around 250 iteration) to learn the best k index and converge. Whereas, In case of the Impicit Quantile agents tries to find the best suitable indexes throughout the 500 iterations and till 500 iterations they are not stable and not able to converge as they are still exploring for the best possible combinations of indexes. Hence, from this experiment we can observe that when the numbers of indexes are k: Rainbow agents are the faster to converge then in comparison to DQN agents whereas, there is a chance that not all agents converge like Implicit Quantile agents. The latter can be due to hyper-parameter settings, internal code settings or to the model itself. Future studies should consider this.

2) For k-offset=+2: Now, by increasing the number of indexes to k+2 we observe the behaviour of different agents. Starting with DQN they reach a reward of 100 till the 250th number of iteration but it is not stable there and eventually the reward drops as we increase the number of iterations to 450 (Fig. 7), we observe this drop because the agent is still exploring and the indexes it got initially were not optimal. In Fig.8 the Implicit Quantile agents shows that are also not able to get a reward of 100 they are fluctuating till the 500 iteration to find the best set of k+2 indexes but till 500th iteration they have not been able to converge. Unlike the other two agents the rainbow agents as shown in Fig.9 converges, it gets a reward of 100 till 150th iteration. Thus for our experiment i.e. k+2 we observe that the agent Rainbow converged whereas the agents DQN and Implicit Quantile did not converge till 500th iteration. Once again, to determine the exact cause of these factors requires further studies.

3) For k-offset=-2: In this experiment, we observe the behaviour of different agents by decreasing the number of indexes to k-2. As shown in Fig.10, DQN agents are trying to get the best possible set of k-2 indexes and they show a fluctuating reward as the agents are still trying to get the best k-2 indexes. The Impicit Quantile agents step rise in the reward to 100 after the 400th iteration (Fig. 11) but again drops as it is still exploring the different combination of k-2 indexes. Whereas, the Rainbow agents in Fig.12 shows a maximum reward around the 200th iteration and it stays there till 450th iteration so this means that it got the best k-2 indexes around the 200th iteration itself and stays till the 450 iterations and beyond. So for k-2 indexes Rainbow again converges whereas the other two agents did not converge.

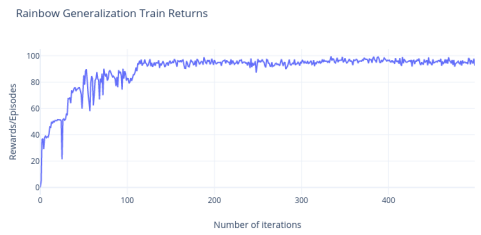

Fig. 13: Rainbow-Generalization Train Returns

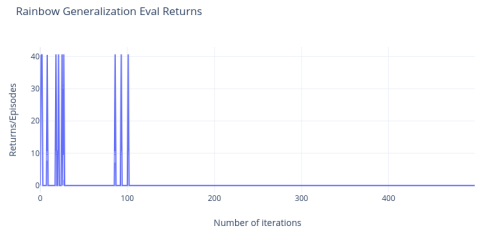

Fig. 14: Rainbow-Generalization Evaluation Returns

B. Experiment 2

In this experiment, we are splitting the queries for training and testing. During the training phase, 5 TPC-H queries are passed to the database. The agents learn with the help of these 5 queries. We are validating the agents learning by testing it on the remaining 2 queries. This experiment aims at the generalization of our agents, answering our third research question.

Our results show that generalization is challenging. Although, we see that agents are training well, which shows that their behavior on the learned task is improving; unfortunately this does not map to the evaluation task. These findings are likely due to the limited kind of training data provided, and to the limited number of iterations. Further work is required to understand these factors better.

C. Experiment 3

This experiment validates our agents learning. Here, we use the model generated in Experiment 1 and compare it against the index advisor tool($pg\_idx\_advisor$). The execution time for indexing the database by the model is compared against that of the the execution time for indexing the database by the index advisor tool. This comparison gives us the efficiency of our model with respect to indexing the database. This answers to the last of our research questions.

TABLE I: Experiment 3

Agent	Time in seconds
Index Advisor	2.994
DQN	0.1877534162
IQN	0.225753615
Rainbow	0.2629348551

Table I shows the time taken by the agents and index advisor. From this table can we state that the time taken by the agents is approximately 10 times less than the time taken by the index advisor to suggest indexes. Hence by this experiment we can state that the agents trained in Experiment 1 are notably faster than the state-of-the-art index advisor tool.

D. Experiment 4

In this experiment, we compare the cost of running a query under different conditions: In a database without indexes (baseline), in a database with indexes suggested by index advisor tool (Semi-automated) and in a database with indexes suggested by deep reinforcement learning agents (No DBA). This complements our evaluation of the first research question.

TABLE II: Experiment 4

Agent	Cost of the Agent
Without Indexes	1550074.43
Index Advisor	1401424.63
DQN	1401424.63
IQN	1467847.742
Rainbow	1401424.63

From the results mentioned in table II we can state that the cost of indexes from the agents are at par with the ones that generated by the index advisory.

With this we conclude our evaluation. In the next section we present further related work which provides further context to our study.

V. Related Work

Over the last two decades, there has been a lot of literature published with respect to the performance improvement of the systems. In this section, we will briefly evaluate the major work done in this field of database system. Prior work in selecting the optimal indexes for overall improvement that takes into account can be split up into the following categories: (1)Identifying the optimal set of indexes for a given workload (2) Using various index advisor tools for recommendation of indexes and (3) Evaluating the systems using possible indexes over different input parameters.

The problem of predictability to see whether the difference between estimated improvement and the actual improvement using number of input parameters for evaluation has been discussed by Borovica et. al.[BAA12]. On the other hand, an alternate approach has been taken by Sharma et. al. [SSD18], where they have introduced NoDBA approach using the machine learning methods to get the optimal set of indexes that will help in the performance enhancement of the systems which was tested on just 5 queries. Further developments included a tool introduced by [CN97] which was created to obtain the indexes but it was done only on a single database system which resulted in increasing the performance of the system by a factor of 4 to 10 without compromising the quality of index selected.

Overall, with all these approaches there was a general overview captured by us which formed a concrete baseline

on what is index selection, how the indexes are selected for a workload and why it is important in database systems.

VI. Conclusions and Future Work

The work in this project suggests, that using DRL agents for index selection is feasible. This is ensured by the fact, that TPC-H benchmark data and queries are being used to evaluate the agents under study. Although, the agents show a variation in their behaviour and are not perfect, we can still observe in Experiment 1 that the agents are able to progress towards learning the optima. Further tests to understand this could be useful and may help in perfecting this study. In addition to this, in Experiment 2, it is clear that the agents do not generalize at all. One of the possible reasons for this can be the adapted design strategy for the this task, which demands further investigation. Considering other aspects apart from the training, agents are shown to be quite competitive in their runtime to give the index recommendations, when compared to state of the art index advisory tools.

Future work involves, aspects that were left as good to study, such as hyper-parameter tuning, adaptation of Ray and Horizon frameworks and incorporating the Learning from Demonstration concept into our environment.

References

[BAA12] Renata Borovica, Ioannis Alagiannis, and Anastasia Ailamaki. "Automated physical designers: what you see is (not) what you get". In: *Proceedings of the Fifth International Workshop on Testing Database Systems*. ACM. 2012, p. 9.

[Cas+18] Pablo Samuel Castro et al. "Dopamine: A research framework for deep reinforcement learning". In: *arXiv preprint arXiv:1812.06110* (2018).

[SSD18] Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. "The case for automatic database administration using deep reinforcement learning". In: *arXiv preprint arXiv:1801.05643* (2018).

[Val+00] Gary Valentin et al. "DB2 advisor: An optimizer smart enough to recommend its own indexes". In: *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*. IEEE. 2000, pp. 101–110.

[FST88] Schkolnick Finkelstein, Mario Schkolnick, and Paolo Tiberio. "Physical database design for relational databases". In: *ACM Transactions on Database Systems (TODS)* 13.1 (1988), pp. 91–128.

[CN97] Surajit Chaudhuri and Vivek R Narasayya. "An efficient, cost-driven index selection tool for Microsoft SQL server". In: *VLDB*. Vol. 97. Citeseer. 1997, pp. 146–155.