8-2021

Deep Learning Data and Indexes in a Database

Vishal Sharma
Utah State University

DEEP LEARNING DATA AND INDEXES IN A DATABASE

by

Vishal Sharma

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Curtis Dyreson, Ph.D. Major Professor	Nicholas Flann, Ph.D. Committee Member
Vladimir Kulyukin, Ph.D. Committee Member	Kevin R. Moon, Ph.D. Committee Member
Haitao Wang, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Interim Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2021

ABSTRACT

Deep Learning Data and Indexes in a Database

by

Vishal Sharma, Doctor of Philosophy

Utah State University, 2021

Major Professor: Curtis Dyreson, Ph.D.
Department: Computer Science

In this thesis, we apply deep learning techniques to databases to solve two specific research problems *1)* index configuration for offline and online workloads and *2)* entity resolution in heterogeneous databases.

Over the last decade, decision-making in a variety of areas has grown enormously towards a data-driven approach. A data-driven application collects and generates a massive amount of data. Such an application require processing data at a fast pace to retrieve information from data, performing Extract-Transform-Load (ETL), and to perform complex analysis. The performance of a data-driven application is significantly affected by a database system. A database system requires tuning for optimal performance. However, there are tens of configuration parameters, it can be challenging to manually configure a database. Moreover, a database has to be reconfigured periodically to keep pace with a changing data collection and workload. The physical design of the database plays an important role in optimal performance. An important component of a physical design is selecting a set of indexes that balances the trade-off between query execution time, storage cost, and maintenance cost. We propose utilizing the query workload history to improve its performance. First, we built an index recommender using deep reinforcement learning for a standalone database. We evaluated the effectiveness of our algorithm by comparing to

several state-of-the-art approaches. Second, we develop a real-time index recommender that can, in real-time, dynamically create and remove indexes for better performance in response to sudden changes in the query workload. Third, we develop a database advisor. Our advisor framework will be able to learn hidden patterns from a workload. It can enhance a query, recommend interesting queries, and summarize a workload.

The entity resolution problem is to match entities from heterogeneous databases. With the emergence of massive amount of data, linking data between data collections, known as entity resolution, has become very important. We build a system to link social media profiles from three popular social media networks. Our system, called LinkSocial , is fast, scalable and accurate.

Overall, this thesis provides evidence for two statements *1) using a historical SQL workload, a database can be optimized autonomously with high accuracy*, *2) using basic features from social media, user entities can be effectively linked across multiple social media platforms.*

(73 pages)

PUBLIC ABSTRACT

Vishal Sharma

A database is used to store and retrieve data, which is a critical component for any software application. Databases requires *configuration* for efficiency, however, there are tens of configuration parameters. It is a challenging task to manually configure a database. Furthermore, a database must be reconfigured on a regular basis to keep up with newer data and workload. The goal of this thesis is to use the query workload history to autonomously configure the database and improve its performance. We achieve proposed work in four stages: (i) we develop an index recommender using deep reinforcement learning for a standalone database. We evaluated the effectiveness of our algorithm by comparing with several state-of-the-art approaches, (ii) we build a real-time index recommender that can, in real-time, dynamically create and remove indexes for better performance in response to sudden changes in the query workload, (iii) we develop a database advisor. Our advisor framework will be able to learn latent patterns from a workload. It is able to enhance a query, recommend interesting queries, and summarize a workload, (iv) we developed LinkSocial , a fast, scalable, and accurate framework to gain deeper insights from heterogeneous data.

To my friends and family

ACKNOWLEDGMENTS

Academically and personally, the journey of PhD has been an excellent learning curve. I would like to thank my advisor Dr. Curtis Dyreson for helping me throughout this journey. I am grateful for his unwavering encouragement and empathy with a beginner like me, as well as his ability to listen, assist, and answer all of my questions. I'd like to express my gratitude for his financial assistance.

I'd like to express my gratitude to Dr. Nicholas Flann, Dr. Vladimir Kulyukin, Dr. Kevin R. Moon, and Dr. Haitao Wang for serving on my graduate committee and for their help and valuable suggestions during my time at Utah State University. I'd also like to express my gratitude to USU's CSE Department for their support. I'd like to express my gratitude to all of my friends and family for their motivation during this journey.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Organizations use databases to manage data; for instance, banks store financial data in databases, marketplaces use databases for tracking products, and scientists deposit knowledge gleaned from experiments in databases. Databases are widely used because they support efficient querying and updating by multiple users.

Traditionally, a machine learning technique was used to acquire patterns from the dataset. A machine learning algorithm learns a mapping from given input data to an output. This approach has limitations concerning the representation of data. The Deep Learning technique solves this problem by learning to represent data and map an input to the output. The learned representation usually outperforms traditional handcrafted features. Deep Learning is also known as representation learning. Deep Learning has been previously used to solve many problems and has been applied to almost every research domain, e.g., Biology, Economics, Engineering, etc. Such representation learning has also gained significant research development in the field of Computer Vision and Natural Language Processing. The research and practice in databases have also been influenced by deep learning. Deep learning has been used for buffer size tuning [1], learning index structure [2], data layout partitioning [3], join order selection [4].

Deep Learning techniques have been previously applied to databases. For instance, it has been applied to tune buffer size tuning [1], learning indexes [2], improve data layout partitioning [3], and reorder joins [4]. A recent survey of literature in the field [5] organized the application of deep learning into four broad categories: *(1)* knob tuning *(2)* optimization *(3)* monitoring, and *(4)* learning physical layout. These categories are depicted in Figure 1.1. Knob tuning the process of adjusting database parameters, such as buffer size, to optimize performance. The problem is complex because a database has many tens of parameters

that can be tuned. Monitoring is keeping track of how the *health* of a database, e.g., throughput, latency, memory use, changes over time, and tuning knobs as needed to ensure continued good health. Optimization is learning how to improve query performance. Query optimization needs to estimate the size of intermediate query results, decide when to use an index to improve a query and explore the space of query execution plans to choose an optimal plan. Finally, physical layout refers to how the data is organized in storage, either on disk or sharded in a cloud computing architecture. In terms of traditional database problems, index selection, buffer size recommendation, and workspace memory allocation are *knob tuning* types of problem. Database performance management, the health of a DBMS, and workload analysis can be categorized as *monitoring* problems. Estimating database cardinality, query cost, and join order selection is classified as *optimization* problems. Learning an index structure, table partitioning, and linking data from sources are categorized as learning a database's *physical layout*.

In this proposal, we focus on three different categories of database improvement using deep learning:

- *Knob Tuning*: We propose an index recommendation framework using Deep Reinforcement Learning for an offline(standalone: RL_Index) and online (streaming: $Indexer++$) workload.
- *Physical Layout*: We propose a LinkSocial framework for an entity linkage using deep learning in a heterogeneous database.
- *Monitoring*: We propose $QueryEnhancer$ a deep learning powered database workload analyzer.

1.1 Outline of Thesis

In the remainder of Chapter 1, we describe the index recommendation problem and related works in more detail in Section 1.2. We then describe Entity Resolution problem in Section 1.3 and propose our overall contribution in Section 1.4. In Chapter 2 we describe

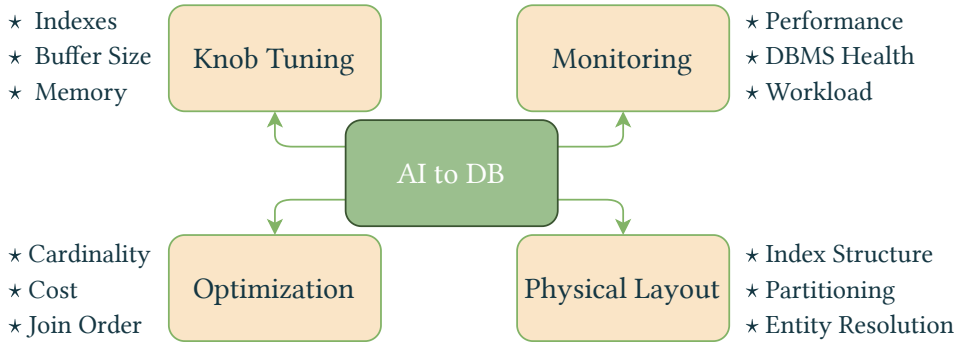

Figure 1.1: Artificial Intelligence applied to different areas of databases

our work on Entity Resolution across social media. In Chapter 3 we describe our solution to index selection for a standalone database. In Chapter 4, we introduce our solution for online index selection. In Chapter **??**, we propose our ongoing work on deep learning powered Query Enhancer.

1.2 Database Index Recommendation

There are three broad approaches to achieving a self-managing database: 1) heuristics, 2) Machine Learning/Deep Learning, and 3) Reinforcement Learning to learn to optimize knobs automatically. A database can be tuned using 1) *external*, and 2) *internal* methods. An *external* method of tuning uses an API to access database management system (DBMS) configurations to control, and an *internal* way is to integrate algorithms for tuning within the DBMS. The former approach for tuning is very widely used because it requires building or modifying (often proprietary) DBMSs. Most of the approaches discussed below are *external* tuning methods, *internal* tuning DB are mostly industry-based, *e.g.,* IBM DB2, Microsoft SQL Server and the Oracle9i optimizers.

One of the focus in this thesis will be on one kind of DBMS knob: *indexes*. A database index is a performance-enhancing data structure. It can be built for a column or several columns of a database table. The only benefit of an index is that a query optimizer can generate a query execution plan that uses the index to speed up the query's evaluation. Only some kinds of queries can be improved with an index. It adds to the total size of

a database. Database indexes are critical to optimizing query workloads. They are very efficient in locating and accessing data. Out of several parameters, database indexes are among the most time-consuming and impacting parameters for a workload performance [6]. In this thesis, one of the problems we address is index recommendation in a database. An approach that is effective and practical in the real world. In the next few sections, we explore previous work, their limitations, and propose our contribution.

1.2.1 Previous Work

Tuning Based on Heuristics

The idea of a self-managing database based on heuristics dates back to the 1970s when a self-adaptive physical database design was proposed [7]. This idea was extended by introducing a heuristic algorithm for index selection [8] and attribute partitioning [9] in a self-adaptive DBMS. In 1992, an adaptive and automatic index selection was proposed based on query workload statistics [10]. Two years later, the *COMFORT* online automatic tuning project was described, it had performance load tuning and a self-tuning buffering method [11]. In the late 1990s, several enhancements were made in Microsoft's SQL server with automatic index selection [12] and physical design tuning wizard [13]. By early 2000, IBM's DB2 was equipped with better API support which led to several new areas of database tuning [14] for instance buffer pool size selection [15], index selection [16], automatic physical database design [17], adaptive self-tuning memory heaps and database memory allocation [18]. Self-tuning architectures were also proposed by Oracle in 9i and 10g DBMS [19, 20].

The methods used to tune DMBS during these years were based on either heuristics (hard-coded rules) or cost-based optimization (greedy approach, genetic algorithms) [21]. Both approaches have limitations, heuristics can not generalize a problem, and cost-based optimization is very cost-ineffective. These techniques are also an offline mode of training where they do not consider previously tuned configuration for the next tuning/online tuning.

Tuning Based on Machine Learning

During the 2010s, Machine Learning techniques have been widely used for optimizing DBMSs. Aken *et al.* [22], utilizes both supervised and unsupervised machine learning for recommending knob configurations. They identify the essential knobs using Lasso, categorize workload using K-means clustering and recommend knob setting in both online and offline settings [23]. Kraska *et al.* [24] proposed SageDB, a new DBMS system capable of learning the structure of data and providing optimal query plans. Recursive model indexes (RMI) [2] learn indexes for efficient data access and perform better than traditional B-tree indexes. They also propose smarter query execution plans to take advantage of GPUs/TPUs for computation. Kossmann and Schlosser [25] proposed a component (workload analyzer, tuner, organizer) based modular framework for self-managing databases. They use cost-effective Linear Programming (LP) algorithms to solve optimization problems. Pavlo *et al.* [26] proposed a self-managing database built from scratch with fully autonomous operation capability. Ding *et al.* [27] used Neural Network for a better query cost estimation and used the estimates for index recommendation. Finally, Neuhaus proposed using a genetic algorithm for optimization and index selection *et al.* [28].

Machine Learning-based tuning has a weakness in that they require enormous data and massive computation power to be effective. Such systems can perform great in a controlled environment, but they have minimal use in real-time scenarios. With these limitations of heuristics and Machine Learning approaches, we move to our third approach, which has not been explored.

Tuning Based on Reinforcement Learning

In recent years, Reinforcement Learning (RL) has become popular technique for *combinatorial optimization*. It has been used in several optimization problems in databases such as optimizing join order [4, 29, 30], query optimization [31–34], self-tuning databases [35, 36] and data partitioning [3, 37, 38]. For the *index selection problem*, Basu *et al.* [39], proposed a tuning strategy using Reinforcement Learning. They formulated index selection as a Markovian Decision Process and used a state-space reduction technique to help scale

their algorithm for larger databases and workloads. Sharma *et al.* [40] proposed *NoDBA* for index selection. They use a given workload ($I_{worload}$) and potential indexes ($I_{indexes}$) stacked as input to the neural network. Specifically, they use Deep Reinforcement Learning with a custom reward function. Their actions include only single indexes that make this approach faster but unable to learn multi-column indexes. Welborn *et al.* [41], introduced latent space representation for workload and action spaces calling it the *Structured Action Space*. This representation enables them to perform other tasks as well, *e.g.,* workload summarization and analyzing query similarity. They use a variant of DQN with dueling called BDQN (Branched Deep Q-Network) for learning index recommendation. Licks *et al.* [42] introduced $SmartIX$ where they use Q-Learning for index recommendation. In their approach, they learn to build indexes over multiple tables in a database. They also evaluate $SmartIX$ using a the metrics $QphH@size$, $power$ and $throughput$. They use $QphH@size$ in their reward function, which makes the evaluation process slow, and in some cases, it may take several days of computation, which impacts the scalability of their approach. We also use the same standard metrics for evaluation but not in our reward function. Kllapi *et al.* [43] propose a linear programming-based index recommendation algorithm. In their approach, they identify and build indexes during idle CPU time to reduce computation cost.

Our literature survey of previous work showed a focus only on building B-tree indexes. A modern query workload has wide diversity, and other index types such as BRIN, Hash, and Spatial indexes are also useful common indexes. Moreover, an index's size also plays a crucial role in database performance, but it has not previously been given much attention. There are no previous approaches for performing an end-to-end, multi-attribute, and multi-type index selection, which is one of our proposal's focus.

1.2.2 Incremental Indexing *Indexer++*

In recent years, cloud-based Software as a service (Saas) has become a prevalent choice for hosting any software application. Such services have several benefits over standalone software applications *e.g.*, scalability, effective collaboration, security, and cost savings. A cloud database provides access to all popular SQL and NoSQL databases as a Saas. A

cloud-based database is usually designed for hundreds or thousands of users. In such an environment database runs through several changes in hardware configuration and workload. A database configuration might be tuned for a general scenario, and it may perform effectively for a certain period in time, but if the trend of queries changes, it will need to be reconfigured. A cloud database needs to be able to adapt to such a change in trends for optimal performance. Currently, a DBA adjusts cloud database parameters; with the increase in the number of parameters and the number of such services, there is a need for automatic configuration. With the proliferation of such cloud-based databases, there is a problem of database adaptation to the streaming workload, requiring attention.

In this research project, we focus on searching for optimal parameters of a database for streaming workload. Overall, our goal is to adapt a database for streaming workload using incremental index recommendation in real-time we call it *Indexer++*. A traditional index selector works on heuristics [44] and in some cases need a DBA for good quality index selection [45]. An ML-based index selection works in batches of streaming queries [46, 47]. Considering the cost of training computation and time, such a recommendation system could be slow and may not be feasible for a real-time scenario. Much practical approach has been proposed using Reinforcement Learning [48–50], where authors utilize Deep Reinforcement Learning (DRL) for online index recommendation.

The area of incremental tuning using Deep Reinforcement Learning(RL) has not gained much attention. The primary reason for that is that learning indexes for a query stream are expensive and time-consuming. The creation and deletion of indexes on an extensive database may take up to several minutes. During this time, a database may not be able to serve any request. With that in mind, if the rendering of database indexes is transferred to the main memory using hypothetical indexes, the online index recommendation can be achieved. This is our focus area of research for this project. We primarily focus on the problem of adapting a database to the changing workload in real-time.

1.2.3 Deep Learning Based *Query Enhancer* (ongoing work)

Further, in this proposal, we focus on SQL Query enhancer. A Query enhancer has

not received significant attention previously. An SQL query is used to retrieve data from a database. An SQL workload represents a historic set of queries requested to a database. Analyzing a workload could facilitate a database, user and DBA in several areas *(1)* Identifying interesting queries, *(2)* Enhancing a query to return interesting information, *(3)* Index recommendation using workload summarization, *(4)* Error prediction, *(5)* Query classification, *(6)* Other administrative tasks. Such workload analytics has been an open research problem. We vision a DB assistant based on Deep Learning, which can analyze and extract patterns from a workload. Firstly, we propose a workload representation using Deep Learning. Such representation will have the capability to perform arithmetic operations. For example, if a query A returns *John* and query B returns *Snow* when we perform $A + B$ should return *John Snow*. Secondly, to measure the workload representation quality, we plan to perform index recommendations using summarized workload and compare with the complete workload. Thirdly, we use this representation to identify interesting queries and enhance queries based on custom metrics.

Previous work on workload summarization based on cost/distance based [51–53], where authors use hard-coded rules and predefined distance measures. Recently, there have been few approaches based on Deep Learning [54–56]. Currently, there are four benchmark datasets for evaluation of SQL workload to vector representation:

- **WikiSQL**: It consists of 26,531 tables extracted from Wikipedia and is the largest benchmark.
- **ATIS/GeoQuery**: ATIS consists of information about flight booking and consists of 25 tables. GeoQuery consists of a USA geographic database with seven tables.
- **MAS**: Microsoft Academic Search (MAS) represents an academic, social network. It consists of 17 tables.
- **Spider**: Consists of 200 tables.

These benchmarks consist of join, grouping, selection, nested, and ordering types of queries. We plan to design and build a state-of-the-art approach that will outperform existing approaches in the datasets mentioned above. Our approach is briefly described in a later section.

1.3 Entity Resolution in Heterogeneous Databases

Entity Resolution is the task of finding the same entity across the different datasets. We primarily focus on the problem of User Profile Linkage (UPL) across multiple social media networks. A social media network can be considered a graph network of user-profiles. Such datasets are heterogeneous. User Profile Linkage (UPL) is the process of linking user profiles across social media platforms. Social media is an amalgam of different platforms covering various aspects of an individual's online life, such as personal, social, professional, and ideological aspects. Social media platforms generate massive amounts of data. Previous studies have analyzed this data to learn a user's behavior, interests, and recommendations. Such studies are limited to using only one aspect of an individual's online life by harvesting data from a single social media platform. By linking profiles from several platforms, it would be possible to construct a much richer body of knowledge about a person and glean better insights about their behavior, social network, and interests, which in turn can help social media providers improve product recommendations, friend suggestions and other services. The UPL is a subproblem of a larger problem that has been studied under different names such as record linkage, entity resolution, profile linkage, data linkage, and duplicate detection. In the next section, we look into previous work related to UPL.

1.3.1 Previous Work

In the field of databases, *entity resolution* links an entity in one table/database to another entity from another table/database, *e.g.,* when linking data from the healthcare to the insurance data [57]. Entity resolution has been referred to as *coreference resolution* [58] in NLP and *named disambiguation* [59] in IR. Approaches to solving the problem fall into three categories: numerical, rule, and workflow-based [60]. Numerical approaches

use weighted sums of calculated features to find similarities. Rule-based approaches match using a threshold on a rule for each feature. Workflow-based approaches use iterative feature comparisons to match.

There have been several approaches that utilize user behavior to solve the pair-wise matching problem, such as a model-based approach [61], a probabilistic approach [62], a clustering approach [63], a behavioral approach [64], user-generated content [65], and both supervised [66] and unsupervised learning approaches [67]. The problem of user linkage on social media was formalized by Zefarani et al. [68] where they used usernames to identify the corresponding users in a different social community. In our LinkSocial framework, we used a supervised approach and mitigated the cost by reducing the number of comparisons.

Most previous work in UPL focuses on pair-wise matching due to challenges in computational cost and data collection. In pair-wise matching, Qiang Ma et al. [69] approached the problem by deriving tokens from features in a profile and used regression for prediction; R. Zefarani et al. [70] used username as a feature and engineered several other features by applying a supervised approach to the problem. Unlike these approaches, LinkSocial can perform multi-platform matching as well as pair-wise matching. Multi-platform UPL has received less attention. Xin et al. [71] approached the multi-platform UPL using latent user space modeling, Silvestri et al. [72] uses attributes, platform services, and matching strategies to link users on Github, Twitter, and StackOverFlow; Gianluca et al. [73] leverage network topology for matching profiles across n social media. Liu et al. [61] use heterogeneous user behavior (user attributes, content, behavior, network topology) for multi-platform UPL but gaining access to such data is not a trivial task.

We propose a scalable, efficient, accurate framework called LinkSocial , for linking user profiles on multiple platforms. LinkSocial collects profiles from Google+, Twitter, and Instagram. The data is cleaned, and features from the data are extracted. The similarity is measured in various ways, depending on the feature. Preliminary matches are then refined, and a final match prediction is made.

1.4 Overall Contributions

- **Offline index recommendation in a database using reinforcement learning**
- **Real-time online database index tuning based on reinforcement learning**
- **A deep learning powered database query enhancer**
- **An entity resolution algorithm for heterogeneous databases**

1.4.1 Papers published:

1. V. Sharma, K. Lee, C. Dyreson, "Popularity vs Quality: Analyzing and Predicting Success of Highly Rated Crowdfunded Projects on Amazon", Computing, 2021

2. V. Sharma, C. Dyreson, "COVID-19 Screening Using Residual Attention Network an Artificial Intelligence Approach", 19^{th} IEEE International Conference on Machine Learning and Applications (ICMLA), Miami, 2020

3. A. Maheshwari, C. Dyreson, J. Reeve, V. Sharma, A. Whaley, "Automating and Analyzing Whole-Farm Carbon Models", 7^{th} IEEE International Conference on Data Science and Advanced Analytics (DSAA), Sydney, 2020

4. V. Sharma, C. Dyreson, "LINKSOCIAL: Linking User Profiles Across Multiple Social Media Platforms", 12^{th} IEEE International Conference on Big Knowledge (ICBK), Singapore, 2018

5. V. Sharma, K. Lee, "Predicting Highly Rated Crowdfunded Products", 9^{th} IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), Barcelona, 2018

1.4.2 Papers under review/preparation:

6. V. Sharma, C. Dyreson and N. Flann, "*MANTIS*: Multiple Type and Attribute Index Selection using Deep Reinforcement Learning", 25^{th} European Conference on Advances in Databases and Information Systems (ADBIS), 2021

7. V. Sharma, C. Dyreson, "*Indexer++*: Workload-Aware Online Index Tuning Using Deep Reinforcement Learning", 2021

8. V.Sharma, C. Dyreson, "The Polyverse Paradigm: Metadata-Aware Programming for Database Applications", 2021

9. V. Sharma, C. Dyreson, "Query Enhancer: Contextual analysis of SQL queries using transformers", 2021

CHAPTER 2

LINKSOCIAL: Linking User Profile Across Multiple Social Media Platforms

LINKSOCIAL: Linking User Profiles Across Multiple Social Media Platforms

Vishal Sharma
Department of Computer Science
Utah State University, Logan, Utah, USA
vishal.sharma@usu.edu

Curtis Dyreson
Department of Computer Science
Utah State University, Logan, Utah, USA
curtis.dyreson@usu.edu

***Abstract*—Social media connects individuals to on-line communities through a variety of platforms, which are partially funded by commercial marketing and product advertisements. A recent study reported that 92% of businesses rated social media marketing as very important. Accurately linking the identity of users across various social media platforms has several applications viz. marketing strategy, friend suggestions, multi platform user behavior, information verification etc. We propose LINKSOCIAL, a large-scale, scalable, and efficient system to link social media profiles. Unlike most previous research that focuses mostly on pair-wise linking (*e.g.,* Facebook profiles paired to Twitter profiles), we focus on linking across multiple social media platforms. LINKSOCIAL has three steps: *(1)* extract features from user profiles and build a cost function, *(2)* use Stochastic Gradient Descent to calculate feature weights, and *(3)* perform pair-wise and multi-platform linking of user profiles. To reduce the cost of computation, LINKSOCIAL uses clustering to perform candidate-pair selection. Our experiments show that LINKSOCIAL predicts with 92% accuracy on pair-wise and 74% on multi-platform linking of three well-known social media platforms. Data used in our approach will be available at http://vishalshar.github.io/data/.**

***Index Terms*—Social Media Analysis, User Profile Linkage, Social Media Profile Linkage, Entity Resolution**

I. INTRODUCTION

Social media is an amalgam of different platforms covering various aspects of an individual's on-line life, such as personal, social, professional, and ideological aspects. For instance, an individual may share professional content on LinkedIn, social pictures on Instagram, and ideas and opinions on Twitter [16]. A recent study found that more than 42% of the adults use more than two social media platforms in everyday life[1].

An individual creates a *profile* to participate in a social media platform. A profile has a *public* view and a *private* view, *e.g.,* a credit card number would be part of the private view. In this paper we are only concerned with a publically available information of a profile that consists of a *username*, *name*, *bio* and *profile image*. This limited profile is at the intersection of the kinds of information in public profiles across social media platforms. An individual has a separate profile for each social media platform.

Social media platforms generate massive amounts of data. Previous studies have analyzed this data to learn a user's behavior [17], interests [18] and recommendations [19]. But such studies were limited to using only one aspect of an individual's on-line life by harvesting data from a single social media platform. By linking profiles from several platforms it would be possible to construct a much richer body of knowledge about a person and glean better insights about their behavior, social network, and interests, which in turn can help social media providers improve product recommendations, friend suggestions and other services.

User Profile Linkage (UPL) is the process of linking user profiles across social media platforms. Previous research has shown how to use features in a profile to achieve UPL. For instance, 59% of the users prefer to keep their *username* the same across multiple social media platforms [13], which makes the *username* an important feature in UPL. But exploiting such features is not straightforward as there can be inconsistent, missing, or false information between profiles. UPL is also computationally expensive, making it difficult to obtain high accuracy in the linkage across platform [32].

We propose a scalable, efficient, accurate framework called LINKSOCIAL, for linking user profiles on multiple platforms. The framework is depicted in Figure 1. To the left of the figure, LINKSOCIAL collects profiles from Google+, Twitter, and Instagram. The data is cleaned and features of the data are extracted. Next, the similarity is measured in various ways, depending on the feature. Preliminary matches are then refined, and a final match prediction is made. Our empirical evaluation shows the efficacy of LINKSOCIAL.

This paper makes three major contributions.

1) We describe how to engineer relevant features for linking user profiles across multiple social media platforms. We show that highly accurate linkage can be achieved by using relatively few public features.

2) We show how to decrease the high computation cost of UPL by using clustering. Our intuition is that if we can reduce the number of linkage attempts, then the cost will decrease, so we focus on pruning low similarity linkages. Given a user's profile from one social media platform, to find similar profiles from other platforms, we cluster candidate profiles using similarity based on bi-grams of the *username* and *name*. Our experiments show that optimization preserve accuracy while reducing computation by 90%. The cost reduction claimed does not include pre-computing cost of Jaccard similarity.

3) We empirically evaluate the effectiveness of our framework

[1]http://bit.ly/2FiRy8i

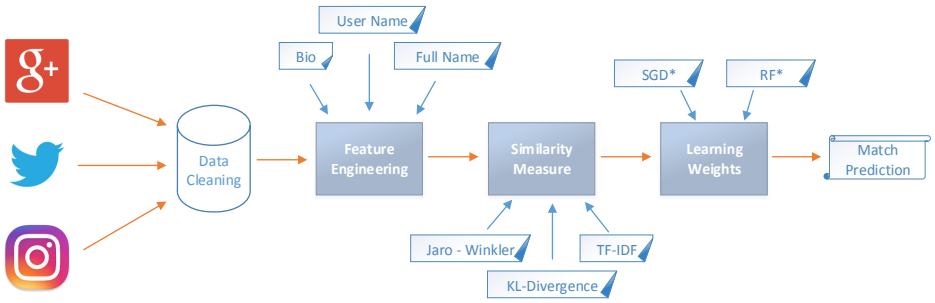

Fig. 1. LINKSOCIAL Framework

by performing extensive experiments and achieving 92% accuracy on pair-wise and 74% accuracy on multi-platform linking.

This paper is organized as follows. Section II motivates the User Profile Linkage problem. In Section III the data acquisition is described. The LINKSOCIAL framework is introduced in Section IV. The section also describes the engineered features, how we measured similarity, and how we optimized for feature weights and reduced the cost of the computation. Section V reports the results of our empirical evaluation of LINKSOCIAL. Related work is presented in Section VI. Section VII presents conclusions and future work.

II. MOTIVATION

In this section, we describe applications of LINKSOCIAL and challenges in UPL.

A. *Applications of* LINKSOCIAL

Data about an individual's social, professional, personal and ideological aspects can be used in various ways.

Security - Social media is widely used for spreading malicious content [21]. Consider a user spreading such content on a social media platform, their activity can be observed on other platforms using LINKSOCIAL. This can help security agencies identify threats or other malicious activity.

Multi-Platform User Behavior - User behavior and activities have been studied extensively using single social media platforms [17]. Linking behaviors from multiple platforms can create a comprehensive picture of a user's behavior. For example, a user A may be active in social life but disassociate professionally. Understanding multi-platform user behavior may lead to insights into why and how friends network differ across platforms. LINKSOCIAL can help link different behaviors to support multi-platform studies.

Information Verification - A user profile could contain false information. For instance, the mobile social networking app Skout reported that, on average, every two weeks three adults masqueraded as teenagers in its forum [22]. By linking user accounts from multiple social media platforms we can check consistency and improve verification of information by merging and validating information from several sources.

Recommendation - Recommendation of products and services is usually based on data from a single social media platform [19]. Data from multiple platforms can enhance the quality and relevance of recommendations, thus, increases user engagement. Most friend recommendation algorithms leverage mutual friends. As pointed out by Shu. et al, multi-platform friend recommendations could improve on pair-wise recommendations [20].

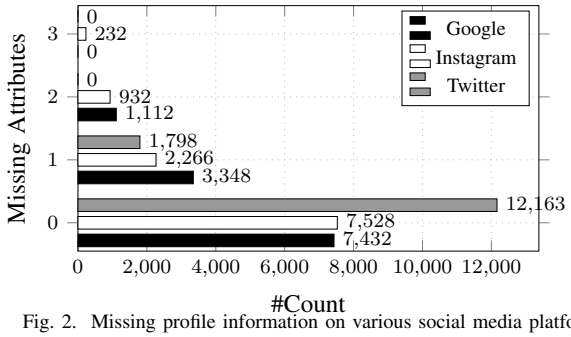

Fig. 2. Missing profile information on various social media platforms.

B. *Challenges in linking users cross social media platforms*

Data Collection - Gathering profiles from social media platforms is not trivial since user privacy concerns limit the available information [27]. Even if we could scrape platforms and collect millions of profiles, ground truthing a UPL solution is elusive since there is no independent way to verify that profiles from different platforms belong to the same user.

Incomplete information - The attributes in a public profile differs across platforms. Some platforms may have an e-mail address, provide a location, or contact information, but most do not. Users also incompletely fill out a profile. Some users may expose gender, age, or ethnicity, but not all users due to privacy. The inconsistency of profile attributes between platforms and among users decreases the potential for linkage.

False information - Faking identity on social media is common [23] as is sharing false content [25] and providing false information about an individual [24]. Social media platforms do not provide verification mechanisms for profile data.

Missing information - A profile provides only a small amount of data about a user since privacy concerns limit the amount of public sharing [26]. Profiles with missing data further exacerbates the difficulty of linking as shown in Figure 2. The paucity of available data coupled with the high rate of missing data complicates the task of accurately linking profiles.

Limited Access - Data from major social media platforms can be accessed through a platform-specific API but due to privacy concerns, social media providers reveal only a limited

amount of data. Even after collecting data, we might lack enough common features to link profiles.

III. Problem Definition

This section gives the problem statement, and discusses data collection and pre-processing.

A. Problem Statement.

Let P_i^k be the i^{th} public profile on social media platform k. Let I be an identify function that maps a P_i^k to the identity of the user who created the profile. For linking profiles from n social media platforms, we use the following objective function.

$$\Phi_n(P_i^1, \ldots, P_k^n) = \begin{cases} 1 & \text{if } I(P_i^1) = \ldots = I(P_k^n) \\ 0 & otherwise \end{cases} \quad (1)$$

Our goal is to build and learn functions $\Phi_2(.)$ and $\Phi_3(.)$ for linking pair-wise and three-platform profiles. We assume that in our dataset, every user has exactly one profile in each social media platform.

B. Dataset Collection

For testing a UPL solution, we need ground truth data. While social media platforms have APIs to access user data, there is no platform to help us link a profile to an individual. However, there are ways that we could build a set of ground truths. For instance, we could crowd-source the ground-truthing (viz. Amazon Mechanical Turks) or use surveys [22], but these methods are prone to getting unreliable data. Instead, we used a novel resource, the website `about.me`, which is a site that requires users to input links to profiles on other social media platforms . When creating a profile on the site, a user will provide links to their other social media profiles.

We used a dataset of 15,298 *usernames* from six social media platforms: Google+, Instagram, Tumblr, Twitter, Youtube, and Flickr [12]. We narrowed the dataset to Instagram, Twitter, and Google+ for our study since they make the *username*, *name*, *bio*, and *profile image* are publicly available.

Next, we built a crawler to collect profiles from the three platforms. Table II displays information about our collected data. We gathered data on 7,729 users from all three platforms, 6,039 users have data available on a pair of platforms and 1,530 have data available only from one platform. Missing profiles could be because of users deactivating their accounts.

C. Dataset Analysis

We analyzed the collected profiles to determine how much information was missing. Figure 2 shows the count of missing profile attributes for each platform. In Google, Twitter, and Instagram there were 28%, 13% and 21% of user profiles with at least one missing attributes, respectively. 9% and 8% of Google and Instagram profiles had at least two missing attributes. There were 87%, 62%, and 69% of profiles on Twitter, Google, and Instagram, respectively, without missing information. Three attributes were missing from 2% of Instagram profiles, making it impossible to match them. The attributes that were presented have some variance. On average, *bios* on Google+ were longer than Instagram and Twitter; 164 characters on Google+ compared to 70 and 96 for Instagram and Twitter. However, the *username* attribute has little variance, it was 11-13 characters on average across all three platforms.

TABLE I
Number of profiles obtained per social media platform.

Social Media	Profile Count
Instagram	10,958
Twitter	13,961
Google+	11,892

TABLE II
Number of users with profiles on pair-wise and multi-platforms

Social Media	Profile Count
Instagram – Google+	614
Twitter – Instagram	2451
Google+ – Twitter	2974
Google+ – Instagram – Twitter	7729

[a]There were 1530 users with only one profile.

IV. The LinkSocial Framework

This section describes LinkSocial, discusses computation cost, and shows how to reduce the cost using clustering.

A. Feature Engineering

LinkSocial uses the basic features in a profile as well as the following engineered features.

Username and Name bi-grams - The *username* is an important feature for profile linkage since people tend to use the same *username* across social media platforms. When a new name is used, parts of the name is often re-used. For example, `John_snow`, `johnsno`, and `snow_john` could belong to the same person. String similarity metrics such as Jaro-Winkler, longest common subsequence, or Levenshtein distance, tend to perform poorly on *name* transpositions [34]. To better align names, we engineer the bi-grams of *usernames* as a feature. We also merge the bi-grams of *usernames* with the bi-grams of names as a feature since people also like to transpose their surname and first name in for a *username*. We engineer the following feature sets.

- bi-gram of *username*. (u_u)
- bi-gram of *name*. (u_n)
- merging above two features. (u_b)

These three feature sets capture a range of different ways to create a *username*.

Character Distribution - User's like to create *usernames* using substrings of their *name* or other personal information (a pet's name or a significant date). To handle scenarios where bi-grams could not capture the similarity, we use the character distribution of *usernames* and *names* as features. To measure distribution similarity we use Kullback–Leibler divergence as defined below,

$$KL_{divergence}(P||Q) = \sum_{i=1}^{n} P(i) \cdot \log \frac{P(i)}{Q(i)} \quad (2)$$

where P and Q are given probability distributions. We engineered the following features sets using character distribution similarity.

- *username*. (u_{u_sim})
- *name*. (u_{n_sim})
- *username + name*. (u_{b_sim})

We perform experiments on real data and have not considered scenarios where *username* and *name* have no relationship.

Profile Picture Similarity - Users often use the same profile picture in multiple platforms. To capture similarity between *profile images*, we use OpenFace [14], an open source image similarity framework. Openface crops images to extract face and uses deep learning to represent the face on a 128-dimensional unit hypersphere. We use ℓ_2 norm to calculate distance between vectors of two *profile images*.

B. Similarity Measures

LINKSOCIAL uses two similarity measures.

Jaro-Winkler Distance - Jaro-Winkler is metric for string matching and is commonly used when matching *names* in UPL [11]. Studies show that the metric performs well for matching string data [15]. Jaro accounts for insertions, deletions, and transpositions while Winkler improves Jaro based on the idea that fewer errors occur early in a *name*.

TF-IDF and Cosine Similarity - LINKSOCIAL uses a different similarity measure for matching profile *bios* since *bios* are longer than *names*. TF-IDF and cosine similarity are widely used for measuring the similarity of documents.

C. Matching Profiles

LINKSOCIAL matches profiles using the basic and engineered features of a profile. We transform the matching problem into a multivariate binary classification problem and optimize it using Stochastic Gradient Descent (SGD). It is defined as follows:

$$h(x) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + ... + w_m \cdot x_m \quad (3)$$

In the equation, $x_1, x_2, ..., x_m$ represents the similarity score of features between two profiles and $w_1, w_2, ..., w_m$ represents their respective weights or coefficient. We use Mean Squared Error (MSE), as our loss function. Considering the predicted values as $h_w(x)_{(i)}$ where $i \in 1, 2, ..., n$ and $y_{(i)}$ as a given value (either a match (1) or no match (0)), we define MSE or cost function $f_{cost}(W)$ as follows:

$$MSE = f_{cost}(W) = \frac{1}{M} \sum_{i=1}^{m} \left(y_{(i)} - h_w(x)_{(i)}\right)^2 \quad (4)$$

LINKSOCIAL uses SGD to optimize the cost function. Partial derivatives of Equation 4 w.r.t to w_1 & w_2 are defined as follows:

$$\bigtriangledown_{w_1} f^{'}_{cost}(w_1) = \frac{1}{M} \sum_{i=1}^{m} -2x_1 \left(y_{(i)} - h_w(x)_{(i)}\right)$$

$$\bigtriangledown_{w_2} f^{'}_{cost}(w_2) = \frac{1}{M} \sum_{i=1}^{m} -2x_2 \left(y_{(i)} - h_w(x)_{(i)}\right)$$

and similarly for other weights.

Derivatives are followed by updating of the weights. For example, w_1 & w_2 as shown below:

$$w_1 = w_1 - \eta \bigtriangledown_{w_1} f^{'}_{cost}(w_1)$$

$$w_2 = w_2 - \eta \bigtriangledown_{w_2} f^{'}_{cost}(w_2)$$

In the above equation, η is the learning rate, a value that typically ranges between 0.0001 - 0.005. During experiments, we also add elastic net regularization for training.

The derivatives and weights are recursively calculated until the equation converges and yields an optimized weight for each attribute based on a training set. To find a match of a given profile we use Equation 5 where we find the profile with maximum score on the given attributes and weights. In Equation 5, W^T is a weight vector calculated using the optimization of Equation 4 and X_u is a vector of attributes. P_i^k is a profile from social media platform k, while U^j is a set of all profiles from platform j.

$$Match(P_i^k, U^j) = max(W^T \cdot X_u), \quad \forall\, u \in U^j \quad (5)$$

Given profile P_i^k, $Match()$ outputs the most similar profile from platform j.

D. Computation Reduction Using Clustering

UPL can be computationally expensive. Given our dataset with 7,729 user profiles, if we are linking pairs of profiles from only two of the platforms, then the number of comparisons will be $7,729 * 7,728 = 59,729,712$. Assuming we can perform 1,000 comparisons per second (which is a very high ballpark), it will require 17 hours to perform UPL. Matching of millions of users across multiple platforms will be infeasible (without the dedication of massive computing power).

To tackle this problem, we introduce *candidate profile clustering*. We can reduce the number of comparisons by pruning low potential comparisons, that is, by avoiding the work of matching profiles that are dissimilar. In our dataset, 45% of the Instagram and Twitter profiles have the same *username* and *name*. By clustering on the bi-gram features for *username* and *name* we can prune comparisons from profiles in different clusters. We have observed in previous studies [35], Jaccard similarity relativly performs well for finding similarity between bi-grams/n-grams and is also computationally not expensive. We rank profiles based on Jaccard similarity and we choose the top 10% of the candidate profiles with the highest score in the cluster. Algorithm 1 gives our approach for building clusters. Our clustering approach is conceptually similar to kNN clustering where distance is defined by Jaccard similarity j_sim and k is 1.

Clustering of profiles before linking can help reduce computation cost significantly but *how big should our clusters be*? If the cluster is big, the computation cost will increase; if cluster is small, we may fail to capture maximum profile matches. To understand the effect of clustering, we plotted cluster size (as a percentage of the total number of profiles) versus match accuracy on our training data as shown in Figure 3. In the plot, the x-axis is the percentage of profiles in a cluster (on

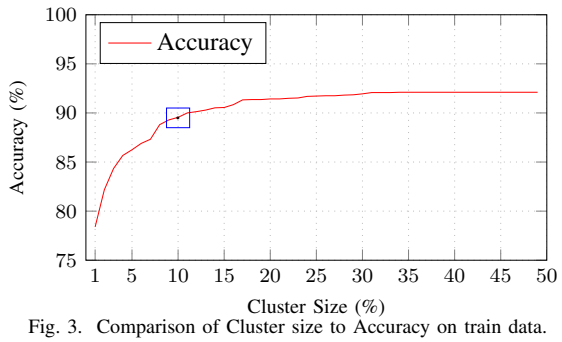

Fig. 3. Comparison of Cluster size to Accuracy on train data.

average). The y-axis shows the maximum accuracy we can achieve since the matching profile has to be in the cluster. We observed that clusters roughly 10% of the size of the data reduces 90% of computation cost while preserving ∼90% of the potential matches.

Algorithm 1 Computing candidate profiles for a given user u

Input: $u, \mathcal{U}$ ▷ $\mathcal{U}$ is set of all users from a different social media than u.

procedure FIND_CANDIDATE_PROFILE($u, \mathcal{U}, n$) ▷ n is number of profile to find in cluster.
  $u_{bg} \leftarrow u_u + u_n$
  for each $p \in \mathcal{U}$ **do** ▷ p is single user profile
    $p_{bg} \leftarrow p_u + p_n$
    $j\_sim \leftarrow Jaccard\_similarity(u_{bg}, p_{bg})$
    $Score[p] \leftarrow j\_sim$ ▷ Insert in dictionary
  end for
  $Candidate\_Profiles = get\_top\_n(n, Score)$
  return $Candidate\_Profiles$ ▷ Returns n candidate profile similar to u
end procedure
procedure JACCARD_SIMILARITY(u_{bg}, p_{bg})
  $Intersection = |u_{bg} \cap p_{bg}|$ ▷ Number of common elements.
  $Union = |u_{bg} \cup p_{bg}|$ ▷ Number of unique elements.
  $Jaccard\_Similarity = \frac{Intersection}{Union}$
  return $Jaccard\_Similarity$
end procedure
procedure GET_TOP_N($n, Score$)
  $Score \leftarrow Sort(Score)$ ▷ Sort Score w.r.t value
  $top\_key = get\_key(Score, n)$ ▷ returns top n key from sorted Score
  for each $key \in top\_key$ **do**
    $top\_n \leftarrow Score[key]$
  end for
  return top_n ▷ return top n candidate profiles
end procedure

V. Experiments

This section reports an experimental evaluation of LinkSocial. We establish a baseline for UPL and verify that LinkSocial can learn Equation 5 and achieve high matching accuracy. We use several variations of calculating weights and compare them with the baseline to determine the impact of the weight calculations. Specifically, we use Random Forest (RF) and Stochastic Gradient Descent (SGD) for calculating weights. We also performed feature analysis to understand which features are important.

TABLE III
Comparison with previous research

Linkage	Authors et. al	Features	Dataset Public	Scalable	Reduce Cost	Scalable Across Platform
Pair-Wise	P. Jain[28]	Private	×	×	×	×
	A. Mal[8]	Public	×	✓	✓	×
	R. Zafa[22]	Public	×	×	×	×
	Y. Li[31]	Private	×	×	×	×
	LinkSocial	Public	✓	✓	✓	✓
Across	X. Mu[32]	Private	×	×	×	✓
	S. Liu[6]	Private	×	×	×	✓
	LinkSocial	Public	✓	✓	✓	✓

A. Experimental Setup

We used the data described in Section III-B. We chose a 60-40 split in the data for training and testing.

Baseline - We built a baseline using Jaro-Winkler and TF-IDF as discussed in Section IV-B. We use Jaro-Winkler to analyze *username* and *name* similarity, and TF-IDF and cosine similarity for profile bios. To find the match of a user profile, we select a profile with the highest score from Equation 6 where the value of the W vector is 1, considering each feature is equally important and X represents the feature vector.

$$f(W) = W^T \cdot X_u \tag{6}$$

Calculating Weights - To calculate weights for pair-wise linking, we generated all the features discussed in Section IV-A for each pair, which gives us data for correct matches. To generate data for mismatches, we randomly chose pairs equal to the number of correct matches and collected their feature scores.

We used normalized variable importance score from RF and also SGD optimization algorithm as explained in Section IV-C for weights calculation. RF was trained using 10-Fold Cross Validation, it was tuned using grid search, and the square root of the number of features was selected as the number of variables to be randomly sampled as candidates at each split. SGD was also performed on the same dataset to calculate weights with a learning rate η of 0.001, with 1,000 iterations.

Computing Candidate Profile - To compute candidate pairs, we follow the approach described in Algorithm 1. In training, we pre-compute each profile's potential matches. To generate a feature vector we score based on clusters. To make sure we have both positive and negative label samples of equal size, we randomly sample negative label data of the same size as the positive label set and generated samples are used to learn the weight vector.

Multi-Platform Profile Linking - We link users across three platforms similar to how we perform pair-wise linking. Due to the very high computation cost, we were unable to run experiments for linking without first clustering candidate profiles. We performed experiments by adding and removing engineered features. To find a similar profile, we choose a profile in one social media platform and tried to find the matching profiles in the other two platforms. We performed

TABLE IV
LINKSOCIAL PERFORMANCE ON PAIR-WISE UPL

	Social Media Pairs (Accuracy %)		
Experiments	G+≡I	T ≡ I	G+≡T
baseline	55.36%	77.86%	56.86%
Prediction without engineered features and clustering.			
with RF	77.53%	82.08%	77.14%
with SGD	76.61%	82.21%	66.24%
Prediction with clustering, no engineered features.			
with CP & RF	82.62%	83.33%	81.40%
with CP & SGD	82.47%	83.32%	81.19%
Prediction with engineered features, no clustering.			
with RF	86.54%	91.17%	84.56%
with SGD	**86.63%**	**91.68%**	**84.58%**
Prediction with engineered features and clustering.			
with CP & RF	84.85%	87.92%	83.20%
with CP & SGD	**84.91%**	**88.29%**	**83.23%**

this experiment for each platform. During training, we choose a platform and compute its respective candidate profiles from other platforms by building feature vector between candidate profiles and the given profile. We then perform SGD and RF to calculate weights for each feature and we used the calculated weights to find similar profiles.

Previous research comparison - Table III compares our approach to previous approaches in terms of feature selection, data availability and scalability. Previous approaches have used different types of features in their algorithms. A *public feature* is publicly available information (*e.g., bio*, *profile image*, *name*, *user name*). *Private* data such as *location* data is proprietary limiting the feasibility of approaches that rely on such information since they need cooperation from businesses that compete against each other. Making a publicly available benchmark dataset is an important aspect for future work and comparison of approaches. Previous researchers have not shared their data (because of privacy concerns and data sources), where as in our approach we have collected all data as public information and have made it available for future research/comparison. As discussed earlier, UPL is a computationally expensive process and scalability of an approach depends on using methods to reduce computation cost to make the solution practical in a real-life scenario. Also, designing an algorithm to scale for new/several platforms is very important aspect. Only a few approaches in past have used computation cost reduction and are designed to scale for new social media platforms.

Previous work accuracy Table VI reports accuracies in a sampling of previous research in both pair-wise and multi-platform linkage. The accuracy is reported "as is" from the papers, experimental setups and measurements differ across papers, for instance previous work with better accuracy for pair linkage used user generated content data (gaining access to such data is difficult) but in our approach we use only publicly available profile data We observe that LINKSOCIAL is among the leaders in pair-wise UPL and the best at multi-platform UPL.

TABLE V
LINKSOCIAL PERFORMANCE ON MULTI-PLATFORM UPL

	Cross Platform		
Experiments	T→(G+, I)	G+→(T, I)	I→(G+,T)
CP & RF	71.56%	72.50%	73.70%
CP & SGD	**72.95%**	**72.86%**	**74.18%**

*RF−Random Forest, SGD−Stochastic Gradient Descent, CP−Candidate Profiles using Clustering, T−Twitter, G+−Google+, I−Instagram

TABLE VI
REPORTED ACCURACY FROM FEW PREVIOUS WORK.

Linkage	Authors	Accuracy
Pair-Wise	P. Jain et al. [28]	39.0%
Social	A. Malhotra et. al. [8]	64.0%
Platform	R. Zafarani et. al. [22]	93.8%
Linkage	Y. Li et. al. [31]	89.5%
	Our Approach	**91.68%**
Across	X. Mu et. al. [32]	44.00%
Multiple	S. Liu et. al. [6] (reported by [32])	42.00%
Platform	**Our Approach**	**74.18%**

B. Evaluation Metrics

In previous studies, accuracy has been used as a reliable evaluation metric for UPL [28]. Given two profiles from different platforms, the accuracy of such matching can be measured as follows. First, assume the following are known.

- *Total number of correct prediction* **(P)**: Number of correct positive prediction by LINKSOCIAL.
- *Total number of positive sample* **(N)**: Number of positive linked profiles in the dataset.

Then the accuracy can be computed as follows.

$$Accuracy(\%) = \frac{|P|}{|N|} \cdot 100 \tag{7}$$

C. Results

We performed several experiments on pair-wise linking and the results are shown in Table IV and Table V. Specifically, we measured the accuracy of LINKSOCIAL on all possible pairs in our dataset namely, Google+ - Twitter, Google+ - Instagram, and Twitter - Instagram. We also performed experiments with features and weights calculated using RF and SGD. As shown in Table [IV], we started with building a baseline for each pair. We achieved 55%, 78%, 57% for Google+ - Instagram, Twitter - Instagram and Google+ - Twitter respectively. We then performed experiments without using engineered features and clustering. We observed that RF produced more accurate matches than SGD. Next, we added candidate pairs but sill no engineered features. In this case, both RF and SGD performed equally well. We then added engineered features and perform experiments without clustering. We observed that SGD outperformed RF. Finally, we used both engineered features and clustering. SGD again performed better than RF. Overall, weights calculated using SGD proved to be more accurate than RF though in some cases the difference was marginal. In the final stage, we observed reduction of accuracy by adding clustering. This is due to the inconsistent *username* and *name* used by a user, since clustering uses *username* and *name* (in our dataset, out of all

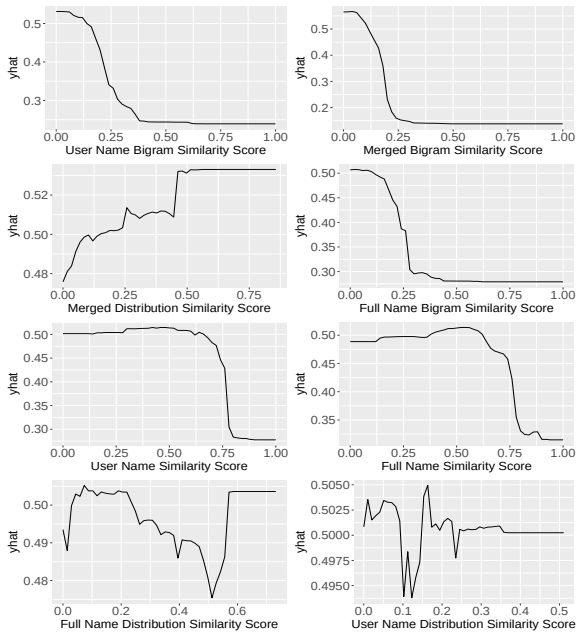

Fig. 4. Partial Dependence Plots for individual features.

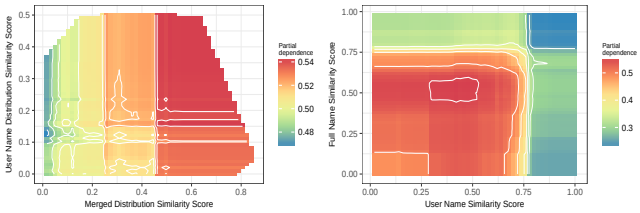

Fig. 5. Partial Dependence Plots for selected pairs of features.

pairs, maximum of 45% users have the same *username* and 22.2% of users with the same *name*), but slight decrease in accuracy could be traded for speed.

We performed several experiments on multi-platform linking and the results are shown in Table V. We observed that, feature weights computation using SGD again outperformed RF. We achieved an accuracy of 73%, 73%, 74% for Twitter→(Google+, Instagram), Google+→(Twitter, Instagram), and Instagram→(Twitter, Google+) respectively. Overall, in our experiments on multi-platform linkage, SGD proved to more accurate.

Model Interpretation - To understand our model, we performed feature analysis using Partial Dependence Plots (PDP) [29]. Given predictor variables and response variables, PDP can interpret the relationship between them. We start by studying the effect of all variables and later choose variable pairs for further study.

In Figure 4, the x-axis represents feature similarity score and the y-axis represents the effect on the class probability. In Figure 4 we observed, that the score from the merged distribution similarity is positively correlated to the model, *username* and *name* similarity contributes to the model until the value of 0.75 then it drops. We plot selected variable pairs, to study their effect on the model with the results show in Figure 5. In Figure 5, the x-axis and y-axis represent the score of respective feature similarity and Partial Dependence represents the marginal effect of features on the class probability. We observed that higher values of merged and *username* distribution similarity score together are highly correlated to the model. Similarly, *username* and *name* similarity score values until 0.75 are highly correlated but the highest values are relatively low. This implies that there are several instances in our data where *username* and *name* similarity scores together are very high (close to 1), but selected profiles do not belong to the same individual. We also observed, with their relatively lower values of similarity, that there are instances where both profiles belong to the same person. Finally, we concluded that the similarity of *username* and *name* are insufficient or unreliable features for linking profiles.

VI. Related Work

UPL is a subproblem of a larger problem that has been studied under different names such as record linkage, entity resolution, profile linkage, data linkage, and duplicate detection. In the field of databases, *entity resolution* links an entity in one table/database to another entity from another table/database, *e.g.,* when linking data from the healthcare to the insurance data [2]. Entity resolution has been referred to as *coreference resolution* [1] in NLP and *named disambiguation* [3] in IR. Approaches to solving the problem fall into three categories: numerical, rule and workflow-based [20]. Numerical approaches use weighted sums of calculated features to find similarity. Rule-based approaches match using a threshold on rule for each feature. Workflow-based approaches use iterative feature comparisons to match.

There have been several approaches that utilize user behavior to solve the pair-wise matching problem, such as, a model based approach [6], a probabilistic approach [5], a clustering approach [7], behavioral approach [28], user generated content [31], and both supervised [9] and unsupervised learning approaches [10]. The problem of user linkage on social media was formalized by Zefarani et al. [13] where they used usernames to identify the corresponding users in different social community. In our framework, we used a supervised approach, and mitigate cost by reducing the number of comparisons.

Most previous work in UPL focuses on pair-wise matching due to challenges in computational cost and data collection. In pair-wise matching, Qiang Ma et al. [4] approached the problem by deriving tokens from features in a profile and used regression for prediction; R. Zefarani et al. [22] used username as a feature and engineered several other features by applying a supervised approach to the problem. Unlike these approaches, LinkSocial can perform multi-platform matching as well as pair-wise matching. Multi-platform UPL has received less attention. Xin et. al [32] approached the multi-platform UPL using latent user space modeling, Silvestri et.al [33] uses attributes, platform services, and matching strategies to link users on Github, Twitter, and StackOverFlow; Gianluca et. al [30] leverages network topology for matching profiles

across n social media. Liu et. al. [6] uses heterogeneous user behavior (user attributes, content, behavior, network topology) for multi-platform UPL but gaining access to such data is not a trivial task.

VII. Conclusion

In this paper, we investigate the problem of User Profile Linkage (UPL) across social media platforms. Multi-platform linkage can provide a richer, more complete foundation for understanding a user's on-line life and can help improve several research studies currently performed only on single social media platform. UPL has many potential applications but is challenging due to the limited, incomplete, and potentially false data on which to link. We proposed a large scale, efficient and scalable solution to UPL which we call LinkSocial. LinkSocial links profiles based on a few core attributes in a public profile: *username*, *name*, *bio* and *profile image*. Our framework consists of *(1)* feature extraction, *(2)* computing feature weights, and *(3)* linking pair-wise and multi-platform user profiles. We performed extensive experiments on LinkSocial using data collected from three popular social media platforms: Google+, Instagram and Twitter. We observed that *username* and *name* alone are an insufficient set of features for achieving highly accurate UPL. UPL is computationally costly, but we showed how to use clustering to reduce the cost without sacrificing accuracy. Candidate profile clustering is based on pruning dissimilar profile comparisons. It reduced 90% of the comparisons which significantly helped in scaling our framework. We evaluate our framework on both pair-wise and multi-platform profile linkage with accuracy 91.68% on pair-wise and 74.18% on multi-platform linkage.

Data about a user from multiple social media platforms has many applications. In future, we plan to *(1)* extend our work to study a user's behavior across platforms, which to our knowledge has not yet been studied, *(2)* add more features to LinkSocial using heterogeneous data, *e.g.,* user content similarity (text, videos, images), network similarity, and patterns across social media platforms, and *(3)* evaluate LinkSocial on more (up to six) social media platforms.

References

[1] J. Cai, M. Strube, "End-to-End Coreference Resolution via Hypergraph Partitioning," COLING, 2010.

[2] S. E. Whang, D. Marmaros, H. Garcia-Molina, "Pay-As-You-Go Entity Resolution," IEEE Transactions on Knowledge and Data Engineering, 2013.

[3] Y. Qian, Y. Hu, J. Cui, Q. Zheng, Z. Nie, "Combining machine learning and human judgment in author disambiguation," CIKM, 2011.

[4] Q. Ma, H. H. Song, S. Muthukrishnan, A. Nucci, "Joining user profiles across online social networks: From the perspective of an adversary," IEEE/ACM ASONAM, 2016.

[5] H. Zhang, M. Kan, Y. Liu, S. Ma, "Online Social Network Profile Linkage," AIRS, 2014.

[6] S. Liu, S. Wang, F. Zhu, J. Zhang, R. Krishnan, "HYDRA: large-scale social identity linkage via heterogeneous behavior modeling," SIGMOD Conference, 2014.

[7] W. W.Cohen, J. Richman, "Learning to match and cluster large high-dimensional data sets for data integration," KDD, 2002.

[8] A. Malhotra, L. C.Totti, W. Meira Jr., P. Kumaraguru, V. A. F. Almeida, "Studying User Footprints in Different Online Social Networks," 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, 2012.

[9] A. Nunes, P. Calado, B. Martins, "Resolving user identities over social networks through supervised learning and rich similarity features," SAC, 2012.

[10] J. Liu, F. Zhang, X. Song, Y. Song, C. Lin, H. Hon, "What's in a name?: an unsupervised approach to link users across communities," WSDM, 2013.

[11] W. E. Winkler, "Overview of record linkage and current research directions," BUREAU OF THE CENSUS, 2006.

[12] B. Lim, D. Lu, T. Chen, M. Kan, "# mytweet via instagram: Exploring user behaviour across multiple social networks," Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on, 2015.

[13] R. Zafarani, H. Liu, "Connecting Corresponding Identities across Communities.," ICWSM, 2009.

[14] T. Baltruvsaitis, P. Robinson, L. Morency, "Openface: an open source facial behavior analysis toolkit," Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on, 2016.

[15] W. Cohen, P. Ravikumar, S. Fienberg, "A comparison of string metrics for matching names and records," Kdd workshop on data cleaning and object consolidation, 2003.

[16] L. Manikonda, V. V. Meduri, S. Kambhampati, "Tweeting the Mind and Instagramming the Heart: Exploring Differentiated Content Sharing on Social Media," ICWSM, 2016.

[17] F. Benevenuto, T. Rodrigues, M. Cha, V. A. F. Almeida, "Characterizing user behavior in online social networks," Internet Measurement Conference, 2009.

[18] P. Bhattacharya, M. BilalZafar, N. Ganguly, S. Ghosh, K. P. Gummadi, "Inferring user interests in the Twitter social network," RecSys, 2014.

[19] M. Jamali, M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," RecSys, 2010.

[20] K. Shu, S. Wang, J. Tang, R. Zafarani, H. Liu, "User Identity Linkage across Online Social Networks: A Review," SIGKDD, 2016.

[21] J. Klausen, "Tweeting the Jihad: Social media networks of Western foreign fighters in Syria and Iraq," Studies in Conflict & Terrorism, 2015.

[22] R. Zafarani, H. Liu, "Connecting users across social media sites: a behavioral-modeling approach," KDD, 2013.

[23] S. C. Herring, S. Kapidzic, "Teens, Gender, and Self-Presentation in Social Media," KDD, 2014.

[24] G. S. O'Keeffe, K. Clarke-Pearson, "The impact of social media on children, adolescents, and families.," Pediatrics, 2011.

[25] D. Miller, E. Costa, N. Haynes, T. McDonald, R. Nicolescu, J. Sinanan, J. Spyer, S. Venkatraman, X. Wang, "How the world changed social media," UCL press, 2016.

[26] R. Gross, A. Aless, "Information revelation and privacy in online social networks," WPES, 2005.

[27] J. M. Kleinberg, "Challenges in mining social network data: processes, privacy, and paradoxes," KDD, 2007.

[28] P. Jain, P. Kumaraguru, A. Joshi, "@I Seek 'Fb.Me': Identifying Users Across Multiple Online Social Networks," WWW, 2013.

[29] A. Goldstein, A. Kapelner, J. Bleich, E. Pitkin, "Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation," journal of Computational and Graphical Statistics, 2015.

[30] G. Quercini, N. Bennacer, M. Ghufran, C. NanaJipmo, "LIAISON: reconciLIAtion of Individuals Profiles Across SOcial Networks," Advances in Knowledge Discovery and Management: Volume 6, 2017.

[31] Y. Li, Z. Zhang, Y. Peng, H. Yin, Q. Xu, "Matching user accounts based on user generated content across social networks," Future Generation Computer Systems, 2018.

[32] X. Mu, F. Zhu, E. Lim, J. Xiao, J. Wang, Z. Zhou, "User Identity Linkage by Latent User Space Modelling," Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[33] G. Silvestri, J. Yang, A. Bozzon, A. Tagarelli, "Linking Accounts across Social Networks: the Case of StackOverflow, Github and Twitter," KDWeb, 2015.

[34] P. Christen "A Comparison of Personal Name Matching: Techniques and Practical Issues," Workshops Proceedings of the 6th IEEE ICDM, 2006.

[35] M. Krieger and D. Ahn "TweetMotif: exploratory search and topic summarization for Twitter," In Proc. of AAAI Conference on Weblogs and Social, 2010

CHAPTER 3

MANTIS: Multiple Type and Attribute Index Selection using Deep Reinforcement Learning

MANTIS: Multiple Type and Attribute Index Selection using Deep Reinforcement Learning

Vishal Sharma, Curtis Dyreson, and Nicholas Flann

Department of Computer Science, Utah State University, Utah, USA
{vishal.sharma, curtis.dyreson, nick.flann}@usu.edu

Abstract. DBMS performance is dependent on many *parameters*, such as index selection, cache size, physical layout, and data partitioning. Some combinations of these parameters can lead to optimal performance for a given workload but selecting an optimal or near-optimal combination is a challenging task, especially for large databases with complex workloads. Among the hundreds of parameters, index selection is arguably the most important parameter for performance. We propose a self-administered framework, called the *Multiple Type aNd Attribute Index Selector* (MANTIS), that automatically selects near optimal indexes. The framework advances the state-of-the-art index selection by considering both multi-attribute and multiple types of indexes within a bounded storage size constraint, a combination not previously addressed. MANTIS combines supervised and reinforcement learning, a Deep Neural Network recommends the type of index for a given workload while a Deep Q-Learning network recommends the multi-attribute aspect. MANTIS is sensitive to storage cost constraints and incorporates noisy rewards in its reward function for better performance. Our experimental evaluation shows that MANTIS outperforms the current state-of-art methods by an average of 9.53% $QphH@size$.

1 Introduction

The performance of a database application is critical to ensuring that the application meets the needs of customers. An application's performance often depends on the speed at which *workloads*, *i.e.,* a sequences of data retrieval and update operations, are evaluated. A database can be *tuned* to improve performance, e.g., by creating an index or increasing the size of the buffer. Figure 1(b) shows the impact of just two configuration parameters, `Memory Size` and `Buffer Size`, on workload performance. We observe that choosing an optimal combination can enhance performance significantly. Currently a database administrator (DBA) manually tunes performance by monitoring performance over time and adjusting parameters as needed. But the growing number of configuration parameters, as shown in Figure 1(a), has increased the complexity of manually tuning performance.

For many queries, *e.g.,* range and lookup queries, a database index significantly reduces query time. An index can be created on a single column or

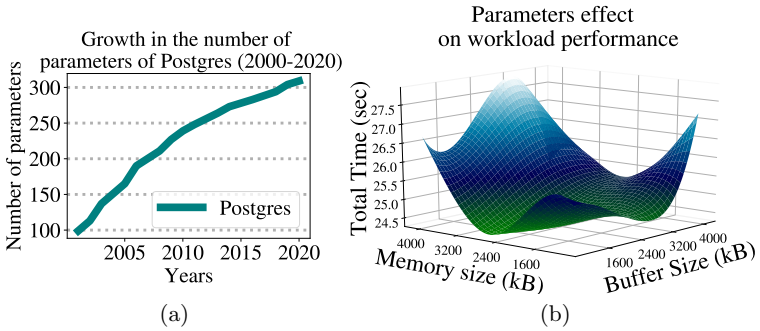

Fig. 1. Figure (a) shows number of parameters in Postgres (2000-2020). Figure (b) shows effect on a workload performance using different parameter settings

several columns of a database table. There are also different *types* of indexes, for instance, Postgres has six index types: B-tree, Hash, GiST, SP-GiST, GIN and BRIN. One possible solution is to simply create all possible indexes for a database, but this approach is infeasible due to large number of potential indexes, *e.g.,* for a single kind of index (B-tree) on a table with N columns, there are $2^N - 1$ potential (multi-column) indexes. There are two other reasons why it is important to limit the number of indexes in a database. First, there are space considerations. An index occupies space in secondary storage that increases the (stored) size of a database. Second, indexes slow down data modification since modifications need to update both the data and the index. Creating too many indexes can decrease the throughput and latency of a database. Hence the set of indexes created for a database should be *parsimonious*, while too few indexes may slow query evaluation, too many may increase space cost and slow down data modification.

The *index recommendation problem* can be defined as finding a set of indices which minimizes the time taken to evaluate a workload and the amount of storage used. Finding a set of indexes which minimizes the cost is a *combinatorial optimization problem*, and adding a disk size constraint makes this problem a *constrained combinatorial optimization problem*. Finding an optimal solution for such a problem is *NP-hard* [19] but Reinforcement Learning (RL) [12] has been used to find approximate solutions for large scale combinatorial optimization problems such as for vehicle routing [17], directed acyclic graph discovery [29], and the traveling salesman problem [14]. RL has also shown that it can learn complex database tasks with a large search space, such as learning optimal parameter configuration for a workload [22].

In this paper, we apply RL to maximize query performance by learning which indexes would be the best to create for a given workload. There has been previous research on index recommendation using Reinforcement Learning [1, 2, 13, 25]. However, previous research has been limited in one of three ways. First, previous research has focused on one type of index, B-Tree, but DBMSs typically support several types,*e.g.,* Postgres has six types. Second, most previous research has in-

vestigated creating only single attribute indexes, but multi-attribute indexes can improve the performance of many queries. Third, previous research has not considered a constraint on storage size, that is, the approaches are *non-parsimonious* and allow the creation of more indexes than needed (there is no penalty for creating too many indexes). A generic framework that captures multi-attribute and different types of indexes is an open research problem. We propose an end-to-end index recommendation system that we call the *Multiple Type aNd Attribute Index Selector* (MANTIS). MANTIS can learn to recommend multi-attribute indexes of different types within a given size constraint.

This paper makes the following contributions.

- We formulate the *Index Recommendation Problem* as a *Markovian Decision Process*. We implement a disk size constraint to limit the total index size in our reward function.
- We propose an end-to-end multi-attribute and multi-index type recommendation framework. Our framework, MANTIS, uses Deep Neural Networks and Deep Q-Learning Networks for recommendations.
- We perform extensive experiments on MANTIS and compare results with current state-of-the-art methodologies.
- We make our code and experimental setup publicly available.

This paper is organized as follows, the next section presents related work. Section 3 gives a precise formulation of the index recommendation problem while Section 4 describes our MANTIS solution to the problem. We present the evaluation of the performance of MANTIS in Sections 5 and 6. Section 7 presents conclusions and future work.

2 Related work

A database can be tuned using either *external* or *internal* methods. An external tuning method uses an API to configure a DBMS while an internal method embeds tuning algorithms in the DBMS. External tuning is widely used because it is generally applicable. Internal tuning needs access to a DBMSs internals, which may be proprietary or dependent on the software architecture of a particular DBMS. Most of the approaches discussed here are external. Internal tuning is primarily industry-based, *e.g.,* the Oracle9i optimizer.

In recent years, Reinforcement Learning (RL) has become a popular external tuning method and has been used to optimize join order [15, 24, 27], in query optimization [7,8,16,18], to self-tune databases [11,28] and to improve data partitioning [3, 5, 26]. For the *index selection problem*, Basu *et al.* [1] proposed a tuning strategy using Reinforcement Learning. They formulate the index selection problem as a Markovian Decision Process and use a state-space reduction technique to scale their algorithm for larger databases and workloads. Sharma *et al.* [22] proposed *NoDBA* for index selection. NoDBA stacks the workload and potential indexes as input to the neural network and uses Deep Reinforcement Learning with a custom reward function for index recommendation. Both of the

above approaches consider only single attribute indexes and a single kind of index. They are unable to recommend multi-attribute indexes. Welborn *et al.* [25] introduced latent space representation for workload and action spaces. This representation enables them to perform other tasks, *e.g.,* workload summarization and analyzing query similarity. They used a variant of DQN with dueling called BDQN (Branched Deep Q-Network) for learning index recommendation. Licks *et al.* [13] introduced *SmartIX* where they use Q-Learning for index recommendation. In their approach, they learn to build indexes over multiple tables in a database. They also evaluate *SmartIX* using the standard metrics *QphH@size*, *power* and *throughput*. They use *QphH@size* in their reward function, which makes the evaluation process slow, and in some cases, it may take several days of computation, which impacts the scalability of their approach. Kllapi *et al.* [6] propose a linear programming-based index recommendation algorithm. Their approach identifies and builds indexes during idle CPU time to maximize CPU utilization without affecting performance. Lan *et al.* [9] propose an index advisory approach using heuristic rules and Deep Reinforcement Learning. Sadri *et al.* [21] utilizes Deep Reinforcement Learning to select indexes for cluster databases.

The above approaches have mostly focused on recommending B-tree indexes. By focusing on a single type of index such approaches lack the capability of utilizing other types of indexes like BRIN or Hash to improve query performance. There are no previous approaches for performing an end-to-end index recommendation for both multi-attribute and multi-type index selection, which is the focus of this paper. Moreover, previous approaches do not support a storage space constraint.

3 Problem Formulation

The index recommendation problem is to select a set of indexes that minimizes the time to evaluate a workload and the amount of storage needed.

A workload W is a set of SQL queries Q_1, Q_2,.., Q_m. An index configuration I is a set of indexes. We calculate the cost of workload evaluation on database D using the cost of the evaluation of each individual query given by $Cost(Q_j, I, D)$. The cost of a workload can be described as follows.

$$Cost(W, I, D) = \sum_{j=1}^{m} Cost(Q_j, I, D)$$

Note that the workload cost does not weight queries differently, though we could trivially include such weights by replicating individual queries in a workload. The index selection problem is to find a set of indexes $I_{optimal}$ that minimizes the total cost of workload $Cost(W, I, D)$ and has a storage cost of at most C.

$$I_{optimal} = \min_{S(I^*) \leq C} Cost(W, I^*, D)$$

In this equation $S(I^*)$ is the total storage space cost of the set of indexes I^*.

4 MANTIS Framework

We designed and built our framework using Deep Neural Networks (DNN) and Deep Q-Learning Networks (DQN). In order to select a suitable set of index types for a workload, our first research goal is *index type selection*. The second research goal is the *index recommendation* to pick possible (single/multiple) attributes for the index. Our framework, research goals, and the challenges we overcome are described in this section.

4.1 Index Type Selection

Almost all DBMSs have several types of indexes. Though the types vary across DBMSs, common index types include *B-tree index* (which includes all B-tree variants and is often the default index type), *block range index* (BRIN) which is helpful in range queries, and *hash index*, which improves the performance of hash joins and point queries. Spatial indexes, such as R-tree indexes, are also commonly available. We trained a Deep Neural Network (DNN) to choose the best types of indexes for a given workload. The DNN models takes workload as an input and predicts potential index types.

DNN model: We convert SQL queries in a workload to a vector representation using feature extraction. Specifically, we extract features describing different query types for B-Tree, BRIN, Hash, and Spatial.

- **feature**_1: Describes the count of each operator used in a query. The operators we search for are [>,<,=,≤,≥]. This feature helps us identify queries searching for equality, range, or general. In case of equality, a Hash index would be preferable, and for queries based on [≤,≥] a BRIN index would be preferred over B-Tree.
- **feature**_2: The number of columns mentioned in a query.
- **feature**_3: The number of conjunctions/disjunctions [‘*and*’,‘*or*’] in a query.
- **feature**_4: To identify spatial queries we extract certain keywords from the query [‘*.geom*’, ‘*.location*’, ‘*distance*’].

Using the above features, we pre-train our DNN model. We use a fully connected multi-layer network with three layers. The first and second layers consist of 32 neurons with *relu* activation, and the output layer consists of *num_classes* (in our case 4) with *sigmoid* activation. The mean squared error (*mse*) is used as the cost function; the number of epochs is 30 and 30% data for validation. The model is trained using *adam* optimizer with a learning rate of 0.001, and we use a learning rate decay (initial rate/epochs) for the stability of the network.

4.2 Index Recommendation

Index recommendation in MANTIS uses Deep Neural Networks for function approximation with the Q-Learning algorithm, also known as the Deep Q-Learning Network (DQN). To build the DQN we formulate the index recommendation

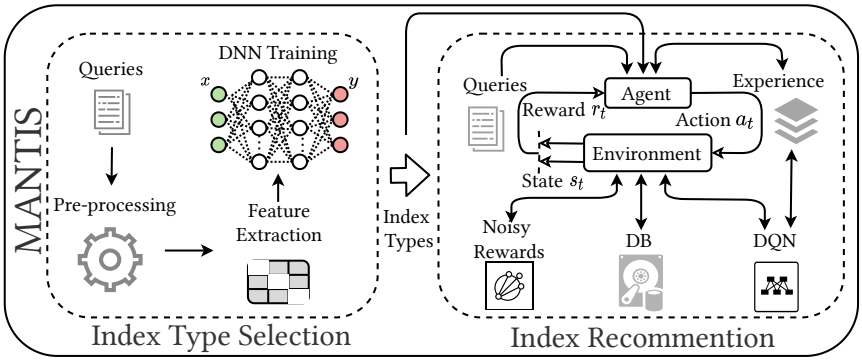

Fig. 2. MANTIS framework

problem as a Markovian decision process (MDP). We extract relevant information from a database to define a *state*, *action*, and a $state_t$ to $action_t$ mapping at time t. We represent a *state* using existing indexes in a system and an *action* using a set of all possible indexes. Both the state and action are deterministic. In a conventional MDP, every action is mapped to a state and an associated reward. The goal of MDP is to reach the final state maximizing cumulative rewards and identifying a *policy*, which is a state-action mapping that selects the appropriate action at a given state. The two fundamental methods to solve MDPs are *value iteration* and *policy iteration*. Value iteration uses the *value* of a state that quantifies amount of future rewards it can generate using the current *policy*, also known as *expected return*. Policy iteration uses the *policy* of a state-action pair that signifies the amount of current reward it can generate with an action at a state using a specific policy.

MANTIS uses a variant of value iteration called Q-Learning. Q-Learning is a value-based, temporal-difference reinforcement learning algorithm. In a Q-Learning algorithm, the agent learns from the history of environment interactions. Q-Learning uses an average of old and newer observations to update neural network. It reduces temporal dependence due to the nature of random data sampling for training, which leads to faster convergence. Our framework can also learn in a constrained environment, in our case, a storage size bound. The constraint is generic and could be used in conjunction with other variables, *e.g.,* buffer size, memory size.

State and Action: An agent interacts with a *state* for the learning process. A *state* is the representation of the environment. In a state representation, the information given is available to the learning agent. It should be capable of accurately explaining the environment. We use a potential set of indexes as the state representation. The *action* space represents all possible actions that can be performed. We calculate the number of possible actions $N_{actions}$ using the

following equation:

$$N_{actions} = (2^{N_{columns}} - 1) \times \textbf{len}(index\_type)$$

where, *index_type* is our DNN model predicting the possible index types to be used and $(2^{N_{columns}} - 1)$ is all possible combination of indexes in a table.

Reward Function: A reward function is another important component of a reinforcement learning agent. We design our reward function based on workload cost estimation. We compare the cost of the index set against the set of all possible indexes as defined below:

$$rewards_{size} = \begin{pmatrix} 1, & index\_size < max\_allowed\_size \\ -1, & otherwise \end{pmatrix}$$

$$r_t = max\left(\frac{index\_cost}{all\_index\_cost} - 1, 0\right) + rewards_{size} \tag{1}$$

where the numerator is the workload cost with a selected set of indexes and the denominator is workload cost with all possible indexes. We also use a reward for the storage size constraint.

Noisy Rewards: Inconsistent rewards can cause an agent's performance to degrade and a learning agent will not be able to maximize rewards. Though we use query cost estimate as a reward, previous research has shown the inefficacy of DBMS cost estimator [10]. To minimize noise in the reward function we perform an 1D convolution filter with a kernel size of five and use the filter in our reward function. Given a input vector f of size k and convolution kernel g of size m a 1D convolution filter can be formulated as follows.

$$(f * g)(j) = \sum_{i=1}^{m} \left(g(j) \cdot f(j - i + m/2)\right) / m$$

We experimented with several, well-known noise-reducing filters, namely Exponential filter, Savitzky-Golay filter, and Kalman Filter. We found that these filters either over or under estimated rewards. The 1D convolution filter also tends to underestimate the cost. However, due to its fast computation time and streaming nature it proved to be a feasible solution. We also added a spike filter to suppress extreme values.

Agent training: The training procedure consists of $N_{episodes}$ episodes, and each episode has N_{index} steps where N_{index} represents the maximum number of indexes. During an episode, the agent performs an action based on a policy and collects rewards for the selected action, this is known as an *experience*. These experiences are stored in a buffer, called the *Replay Buffer*, for sampling and training. The cost of an action is calculated by its effect on the total workload cost of retrieval using the rewards function from Equation 1. The computed rewards are adjusted using the 1D convolution filter. At the end of this step *state*, *action*, *rewards* are returned. During training we use the a configuration batch size of 32, consisting of 100 episodes, a maximum index number of N_{index}:

$3-6$, a priority scale of η: 0.97, a learning rate of α: 0.001, a discount factor of β: 0.97, and a storage bound of 10-20MB. Our complete framework is depicted in Figure 2.

5 Experimental Setup

We perform experiments on two datasets, a standard database benchmark TCP-H [23] and a real time dataset IMDB [10]. We use PostgreSQL as a choice for our database. We create an OpenGym environment for command based database interaction. All experiments were performed on a computer with Intel i7 5820k, Nvidia 1080ti, 32GB of RAM running Ubuntu 18.04 OS. We use Python 3.7 and libraries (powa, gym, psycopg2, sklearn, TensorFlow, Keras) to write and train our framework. The DNN and DQN were trained on Nvidia 1080ti (3584 CUDA cores and 11GB DDR5 RAM) with CUDA and cuDNN configured for performance enhancement.

1. **TPC-H:** TPC is the most well-known and most widely-used family of database benchmarks. TPC-H is the benchmark for decision support systems. We generate a dataset of 120k tuples using TPC-H and randomly generate queries as follows: we randomly select columns and a value from a table. We then randomly select an operator [`>`, `<`, `=`] and predicate [`and`, `or`]. By choosing between one and four columns, we create four different sets of queries (1C, 2C, 3C, 4C). The variety of queries assists in the validation of our framework's efficacy. We generate 100 queries for each TPC-H experiment.
   Few randomly generated queries used in our experiments:

```
1C: SELECT COUNT(*) FROM LINEITEM WHERE L_TAX < 0.02
2C: SELECT COUNT(*) FROM LINEITEM WHERE L_ORDERKEY < 11517219
    OR L_TAX < 0.02
3C: SELECT COUNT(*) FROM LINEITEM WHERE L_SUPPKEY < 18015 AND
    L_PARTKEY > 114249 AND L_TAX > 0.06
4C: SELECT COUNT(*) FROM LINEITEM WHERE L_ORDERKEY = 8782339
    AND L_TAX = 0.01 AND L_PARTKEY = 264524 AND L_SUPPKEY >
    14028
```

2. **IMDB:** IMDB is a large database of movie-related data. There are 21 tables, with a few large tables such as `cast_info` table, which has 36 million records, and the `movie_info` table, which has 15 million records. It has 113 computationally intensive SQL queries with multi-joins. We randomly and equally divide the queries in three stage with 37 (Stage 1), 38 (Stage 2), and 38 (Stage 3) queries. The aim of separating queries is to create more real-time scenarios for framework validation.

5.1 Performance Metric

We measure performance using standard DBMS metrics, such as *Power@size*, *Throughtput@Size*, and *QphH@Size*.

Power@Size tests the durability of the indexes chosen for inclusion and deletion of documents throughout the database. This includes a variety of steps, including (1) a refresh function RF1 that inserts 0.1% of the table's data, (2) the execution of a single stream of queries, and (3) the time taken by the RF2 refresh feature, which deletes 0.1% of the table's records at random. The following equation is used to calculate the metric:

$$Power@Size = \frac{3600}{\sqrt[N_q]{(\alpha_{i=1}^{N_q} ET(i)) \times (\alpha_{j=1}^{2} RF(j))}} \times Scale\,Factor$$

where, N_q is number of queries, $ET(i)$ is execution time for each query i, $RF(j)$ is the time taken by the two refresh functions, and $Scale\,Factor$ is the factor of database size used from TPC-H and IMDB.

Throughput@Size measures the processing capability of a system (disk I/O, CPU speed, memory bandwidth, BUS speed, etc.). It is computed using the equation below:

$$Throughput@Size = \frac{Query\,Stream \times N_q}{Total\,Time} \times 3600 \times Scale\,Factor$$

where, $Query\,Stream$ is the number query streams (for our experiments we used only a single stream) and $Total\,Time$ is the time taken to execute all queries for all streams.

QphH@Size measures multiple aspects of database performance. It measures the query processing power and throughput when queries are from multiple streams. The Query-per-Hour Performance Metric (QphH) is calculated using $Power@Size$ and $Throughput@Size$, as shown below:

$$QphH@Size = \sqrt{Power@Size \times Throughput@Size}$$

Experimental Design: In the TPC-H 1C with only single attribute index selection, there are 16 states (number of columns from `LINEITEM`). We select only 4 columns for multi-attribute index selection and for 2, 3, and 4 column indexes there are 15, 80, and 255 states respectively. For the IMDB dataset, we use all tables and columns in index selection. We use only single column indexes and the state space consists of 108 states. The number of states is crucial for initialization of action and state space of RL agent. The baseline is evaluated on the database with the same records and workload.

5.2 Baselines

We equate our system to the baselines mentioned below. We choose two enterprise-based solutions and two index recommender using RL, which we re-implemented based on information provided in their research papers.

POWA [20]: The PostgresSQL Workload Analyzer is an optimization tool for PostgresSQL. It collects various statistical data from a database and suggests indexes to optimize workload.

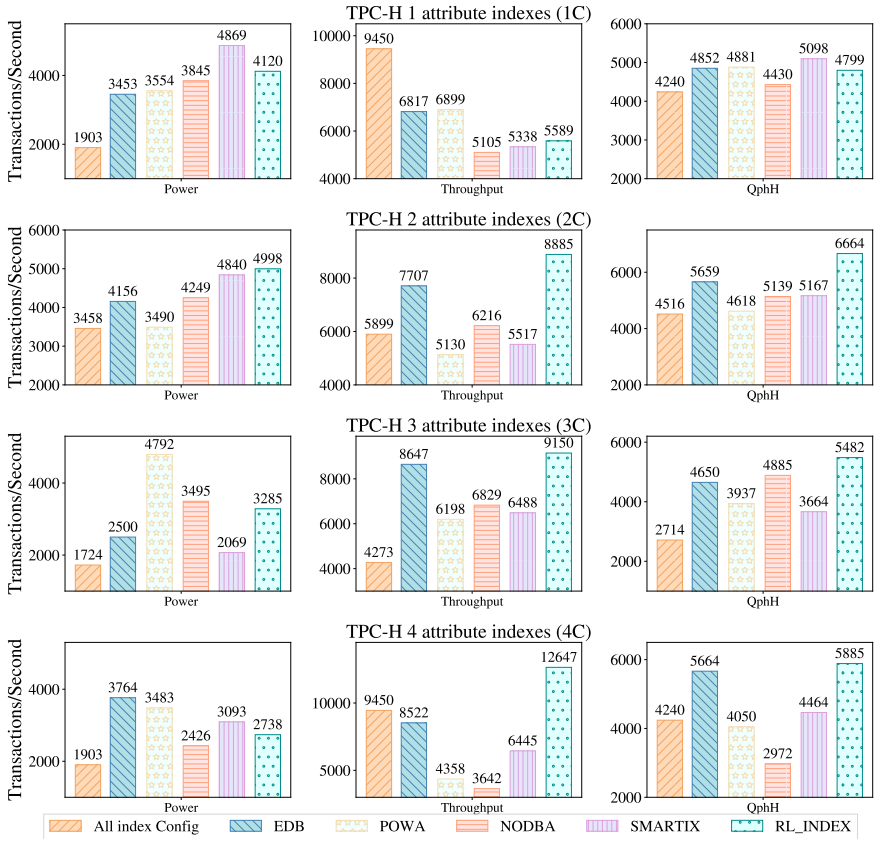

Fig. 3. Power, Throughput and QphH values of index recommendation methods and there comparison in different scenarios on TPC-H dataset.

EDB [4]: EnterpriseDB's Postgres advanced server is designed to customize, tune, and handle massive Postgres database deployments. As a benchmark, we use their index advisor tool.

NODBA [22]: We reimplement NODBA based on details from paper. The authors use DQN for index recommendation. They use query and index configuration as input to the Neural Network.

SMARTIX [13]: We reimplement SMARTIX based on details provided in the paper. The authors use $QphH@Size$ as reward function and Q-Learning for selecting indexes.

All Index: We create all possible indexes for the database and use that as a benchmark. For experiments with IMDB, we do not use **All Index** due to the large index space size and also **NODBA**, it doesn't support multi-table index selection.

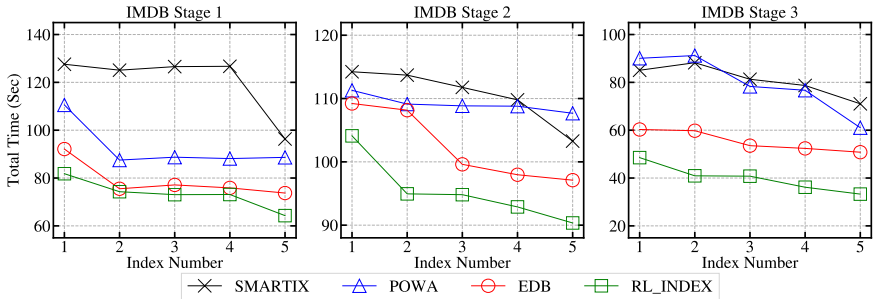

Fig. 4. Workload execution cost with respect to selected indexes on IMDB dataset

6 Results

6.1 TPC-H

* **First Scenario (1C - one column indexes)**: We calculate *power@Size*, *Throughput@Size* and *QphhH@Size* for all of the baseline systems. We observe that most of the baseline and MANTIS performed similarly. Specifically, SMARTIX performs the best in this setting, followed by POWA and EDB. Most of the baselines are designed and tuned specially for single index selection scenarios.
* **2C, 3C and 4C Scenario (multi-attribue indexes)**: We use both one and two columns indexes for 2C. We observe that MANTIS performs best with 17.7% *QphH* improvement to the second baseline. We use 1, 2 and 3 columns index for 3C scenario. Our framework shows 12.1% *QphH* improvement to the second baseline. We use 1,2,3 and 4 columns for 4C scenario. We observe 4% *QphH* improvement of our framework over the best baseline. All the results are shown in Figure 3.

6.2 IMDB

We observe MANTIS outperformed other baseline systems in all three stages, as shown in Figure 5. Specifically, there is 3.19%, 2.9%, and 17.3% of *QphhH* improvement to the best baseline at Stage 1, Stage 2, and Stage 3 respectively. Overall, we observe that our framework outperforms other baselines (3/4) on TPC-H and IMDB (3/3) datasets.

To better understand the results from IMDB, we designed an experiment to evaluate performance. Our objective is to evaluate how successful selected indexes are. We would like to be able to see a drop in the workload's overall execution time once an index is created. We re-ran the benchmark and measured performance after every index creation. The results are shown in Figure 4. The index selected using MANTIS took the least time in all stages. We also observe that the first index selected by MANTIS is optimal in all stages. There is a steady reduction in workload execution costs, which is ideal.

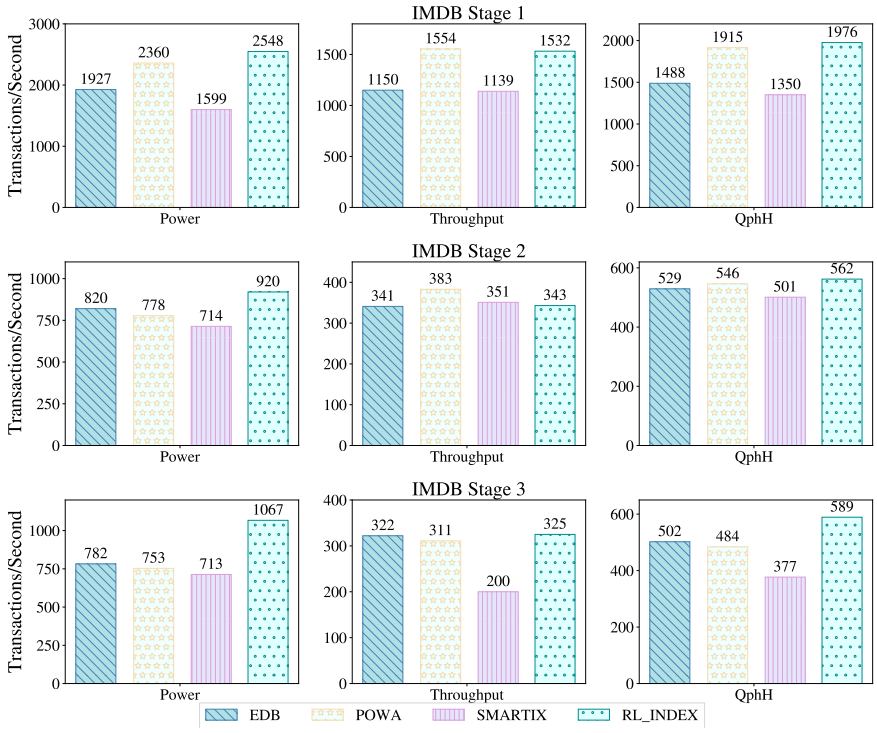

Fig. 5. Power, Throughput and QphH values of index recommendation methods and there comparison in different scenarios on IMDB dataset.

7 Conclusion and Future Work

This paper presents MANTIS, a framework to recommend indexes for enhancing the efficiency of a query workload using Deep Reinforcement Learning. Our implemented framework uses a Deep Neural Network for index type selection and a Deep Q-Learning Network algorithm for multi-attribute index recommendation. In comparison to previous methods, MANTIS can learn and propose single-attribute, multi-attribute, and multi-type indexes. We evaluated MANTIS with four other state-of-the-art methods using two standard benchmark datasets. We use standard DBMS performance metrics *QphH@Size* for evaluation. The experiments show that MANTIS can significantly outperform (6/7 cases) the state-of-the-art index recommendation methods. In future, we plan to extend our work by adding more configuration parameters. We also plan to study temporal index selection for temporal databases, which is an open problem. Finally, an exciting area of future research is to use machine learning to optimally modify or extend a set of indexes in response to a changing workload, in essence performing incremental index selection.

References

1. Basu, D., Lin, Q., Chen, W., Vo, H.T., Yuan, Z., Senellart, P., Bressan, S.: Cost-model oblivious database tuning with reinforcement learning. In: DEXA. p. 253–268. DEXA 2015, Springer-Verlag, Berlin, Heidelberg (2015)
2. Ding, B., Das, S., Marcus, R., Wu, W., Chaudhuri, S., Narasayya, V.R.: AI meets AI: Leveraging query executions to improve index recommendations. In: SIGMOD. p. 1241–1258 (2019)
3. Durand, G.C., Pinnecke, M., Piriyev, R., Mohsen, M., Broneske, D., Saake, G., Sekeran, M.S., Rodriguez, F., Balami, L.: Gridformation: Towards self-driven online data partitioning using reinforcement learning. In: aiDM'18 (2018)
4. EDB: Enterprise Database (EDB): Power to Postgres (2020), https://www.enterprisedb.com/
5. Hilprecht, B., Binnig, C., Röhm, U.: Towards learning a partitioning advisor with deep reinforcement learning. In: Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management. aiDM '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3329859.3329876, https://doi.org/10.1145/3329859.3329876
6. Kllapi, H., Pietri, I., Kantere, V., Ioannidis, Y.E.: Automated management of indexes for dataflow processing engines in iaas clouds. In: EDBT. pp. 169–180 (2020)
7. Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J., Stoica, I.: Learning to optimize join queries with deep reinforcement learning (2018)
8. Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J.M., Stoica, I.: Learning to optimize join queries with deep reinforcement learning. CoRR **abs/1808.03196** (2018), http://arxiv.org/abs/1808.03196
9. Lan, H., Bao, Z., Peng, Y.: An index advisor using deep reinforcement learning. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management. pp. 2105–2108 (2020)
10. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T.: How good are query optimizers, really? Proc. VLDB Endow. **9**(3), 204–215 (Nov 2015). https://doi.org/10.14778/2850583.2850594, http://dx.doi.org/10.14778/2850583.2850594
11. Li, G., Zhou, X., Li, S., Gao, B.: Qtune: A query-aware database tuning system with deep reinforcement learning. Proc. VLDB Endow. **12**(12), 2118–2130 (Aug 2019). https://doi.org/10.14778/3352063.3352129, https://doi.org/10.14778/3352063.3352129
12. Li, Y.: Deep reinforcement learning: An overview. CoRR **abs/1701.07274** (2017), http://arxiv.org/abs/1701.07274
13. Licks, G.P., Couto, J.C., de Fátima Miehe, P., De Paris, R., Ruiz, D.D., Meneguzzi, F.: Smartix: A database indexing agent based on reinforcement learning. Applied Intelligence pp. 1–14 (2020)
14. Ma, Q., Ge, S., He, D., Thaker, D., Drori, I.: Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning (2019)
15. Marcus, R., Papaemmanouil, O.: Deep reinforcement learning for join order enumeration. In: Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management. pp. 3:1–3:4. aiDM'18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3211954.3211957, http://doi.acm.org/10.1145/3211954.3211957

16. Marcus, R., Papaemmanouil, O.: Towards a hands-free query optimizer through deep learning. CoRR **abs/1809.10212** (2018), http://arxiv.org/abs/1809.10212
17. Nazari, M., Oroojlooy, A., Snyder, L., Takác, M.: Reinforcement learning for solving the vehicle routing problem. In: Advances in Neural Information Processing Systems. pp. 9839–9849 (2018)
18. Ortiz, J., Balazinska, M., Gehrke, J., Keerthi, S.S.: Learning state representations for query optimization with deep reinforcement learning. In: Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning. pp. 1–4 (2018)
19. Piatetsky-Shapiro, G.: The optimal selection of secondary indices is np-complete. SIGMOD Rec. **13**(2), 72–75 (Jan 1983)
20. Powa-Team: PostgreSQL Workload Analyzer (Powa) (2020), https://github.com/powa-team/powa
21. Sadri, Z., Gruenwald, L., Lead, E.: Drlindex: Deep reinforcement learning index advisor for a cluster database. In: Proceedings of the 24th Symposium on International Database Engineering & Applications. IDEAS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3410566.3410603, https://doi.org/10.1145/3410566.3410603
22. Sharma, A., Schuhknecht, F.M., Dittrich, J.: The case for automatic database administration using deep reinforcement learning. CoRR **abs/1801.05643** (2018), http://arxiv.org/abs/1801.05643
23. TPC-H: TPC-H Decision Support Benchmark (2020), http://www.tpc.org/tpch/
24. Trummer, I., Moseley, S., Maram, D., Jo, S., Antonakakis, J.: Skinnerdb: Regret-bounded query evaluation via reinforcement learning. Proc. VLDB Endow. **11**(12), 2074–2077 (Aug 2018). https://doi.org/10.14778/3229863.3236263, https://doi.org/10.14778/3229863.3236263
25. Welborn, J., Schaarschmidt, M., Yoneki, E.: Learning index selection with structured action spaces. arXiv preprint arXiv:1909.07440 (2019)
26. Yang, Z., Chandramouli, B., Wang, C., Gehrke, J., Li, Y., Minhas, U.F., Larson, P.r., Kossmann, D., Acharya, R.: Qd-tree: Learning data layouts for big data analytics. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. p. 193–208. SIGMOD '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3318464.3389770, https://doi.org/10.1145/3318464.3389770
27. Yu, X., Li, G., Chai, C., Tang, N.: Reinforcement learning with tree-lstm for join order selection. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE). pp. 1297–1308 (2020)
28. Zhang, J., Liu, Y., Zhou, K., Li, G., Xiao, Z., Cheng, B., Xing, J., Wang, Y., Cheng, T., Liu, L., Ran, M., Li, Z.: An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: Proceedings of the 2019 International Conference on Management of Data. p. 415–432. SIGMOD '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3299869.3300085, https://doi.org/10.1145/3299869.3300085
29. Zhu, S., Ng, I., Chen, Z.: Causal discovery with reinforcement learning. In: International Conference on Learning Representations (ICLR) (2020)

CHAPTER 4

Indexer++: Workload-Aware Online Index Tuning Using Deep Reinforcement Learning

Indexer++: Workload-Aware Online Index Tuning Using Deep Reinforcement Learning

Vishal Sharma
Utah State University
Department of Computer Science
vishal.sharma@usu.edu

Curtis Dyreson
Utah State University
Department of Computer Science
curtis.dyreson@usu.edu

ABSTRACT

The optimal set of indexes is a critical component for a high performance of the Database Management System (DBMS). The Database Administrator (DBA) selects a set of indexes based on historical workload analysis. A modern database processes thousands of queries per minute. A DBA's manual index selection for such a database is a challenging task. Such a modern database is designed to serve multiple users and applications, and a query workload in such a database is heterogeneous in nature. With the increasing workload complexity, the optimal index configuration for such a database also varies. The ability to learn and adapt to evolving workloads is critical for a database performance. This paper proposes an autonomous workload-aware online index selection using Deep Reinforcement Learning. Our approach uses a combination of text analytic techniques and deep reinforcement learning. It consists of two steps *(i)* understanding a workload using embeddings. To support generalized workload analytics, we evaluate several Natural Language Processing (NLP) techniques for workload vector representation. We observe, vector representation using pretrained transformer-based NLP models (BERT, Transformer, XLM, etc.) outperforms existing feature-based approaches. We use NLP models that have been pre-trained to detect changes in workload patterns. Specifically, query workloads are embedded using pretrained models and K-medoids are used to cluster historical and future workload trends. *(ii)* index selection, using deep reinforcement learning. We improve the performance of Deep Q-Networks (DQN) by utilizing disk size constraints and handling noisy rewards in our reward function using a 1D convolution filter. We also introduce *Priority Experience Sweeping* in a DQN for an online index tuning. We perform several experiments using benchmark and real life datasets on our framework. We observe that our approach could effectively detect changes in workload trends and select indexes in a short period of time.

1 INTRODUCTION

An important part of setting up a database is choosing a set of indexes that balances the trade-off between query execution time, storage cost, and update cost [1]. Having too few indexes may slow the evaluation of some queries while too many may increase the cost of data update and the size of the database. Though choosing the right set of indexes is critical to achieving good performance [13] selecting an optimal set of indexes, which we will call the *index selection* problem, is *NP-hard* [47].

Conference'17, July 2017, Washington, DC, USA
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Historically a database administrator chose the set of indexes, but research in self-managing databases has demonstrated that automated index selection can outperform manual selection. Previous research has focused on *offline* automated methods. An offline indexer uses the current state of the database and a representative query workload to determine which indexes to create. Offline indexers use heuristics [11, 18, 44, 57], machine learning [15, 26, 27, 58], or reinforcement learning [3, 24, 53] to recommend a set of indexes.

The set of indexes generated by an offline indexer can become *stale* if the workload or database changes. A stale set can be refreshed by re-running the offline indexer to recreate the indexes. But the cost of offline indexers can be high (on the order of hours of computation time) since an algorithm may run thousands of queries and build many indexes to determine the optimal set of indexes. When offline indexers are run once the cost of running the algorithm is usually a minor concern (the cost is amortized over the lifetime of the database) but frequent periodic runs can degrade performance.

In contrast to offline index selection, the goal of *online* index selection is to *keep the set of indexes fresh.* An online indexer generates a set of indexes initially and then incrementally adapts the set in response to *trends* in workload or database changes. Online indexers face several challenges not shared by their offline counterparts.

(C1) **Noise Resilience**: Online indexers have the problem of how to *identify a trend*, *e.g.,* how many new queries does it take to indicate a trend? An online indexer must be able to adapt to changing trends but avoid short-lived trends since there is a cost to building or removing an index. A constantly shifting set of indexes could worsen DBMS performance.

(C2) **Overhead**: Cost is a critical concern. The tuning process must be able to run concurrently with other DBMS operations without adversely impacting workload processing. Low overhead ensures that an online indexer can (in effect) run continuously to respond to changing trends.

(C3) **Trend Detection**: Trends move fast. An inability to detect them quickly may result in delayed index creation. Too big of a delay in trend detection may, in return, reduce the utility of the indexer.

(C4) **Responsiveness**: The tuning approach should be able to respond quickly, on the order of minutes, if it takes several hours or more for index selection the tuning may miss a trend. Ideally an online indexer will be able to achieve real-time responsiveness.

Previous research in online index selection [7, 39, 46, 48, 50] focused on how to analyze a query workload window to generate a set of indexes. These approaches do not adapt to changing workload patterns. They also have high cost and so have a long delay between

trend detection and index selection. This reduces the utility of having an online index selector.

This paper presents **Indexer++**, a workload-aware, online indexer. The paper makes the following contributions.

- We propose a novel *workload trend detection* technique using a pretrained transformer based on a NLP model and a *K-medoids* clustering algorithm.
- We describe a real-time, online index selection framework that uses Deep Reinforcement Learning to predict trends and tune the set of indexes.
- We use Reinforcement learning to learn the interactions between indexes [29] that can speed up performance [51].
- We present an extensive evaluation of our framework. The experiments demonstrate that framework is able to address challenges C1-C4 listed above.

This paper is organized as follows. Section 2 describes related work and Section 3 formulates the problem. In Section 4 we describe our framework, introduce a workload embedding technique, and use the technique to detect patterns of change in a workload. The section also describes our algorithm for online index selection. Section 5 presents experiments, and finally Section 6 concludes the paper and describes future work.

2 RELATED WORK

This section explores related work on workload embedding and online index tuning in DBMSs.

Word Embedding: In the NLP field, obtaining a vector representation of word (*word embedding*) has been extensively studied. The idea of such representation was introduced by Hinton *et al.* [21]. There has been tremendous growth in the NLP research field with the applications of Neural Network for extracting embeddings. Mikolov *et al.* [42] proposed *word2vec* a neural network based model that can learn latent relationship between words from a corpus of text. This advancement led to solving several challenging problems of NLP with high accuracy such as semantic analogy [33], sentiment analysis [52], document classification [35] to name a few. The idea was later extended for a sentence and document [31]. To improve the vector representation other deep learning techniques were also applied such as CNN [40], RNN [37], LSTM [36]. There was a significant improvement in the quality of word embedding with the introduction of Deep Bidirectional Transformers. Few models based on transforms are Bert [14], Transformer-XL [12], XLM [28]. We use pretrained transformer based models for word embedding in our framework. Our goal of using word embedding is to learn the semantic and syntactic relationship between queries.

Query Embedding: There has been several previous approaches to vectorize an SQL query. Bordawekar *et al.* [5, 6] utilize pre-trained *word2vec* models for query embedding and introduce cognitive capability in databases. Jain *et al.* [23] performs error prediction, and workload summarization using vector representation learnt by training a LSTM based autoencoder. Bandyopadhyay *et al.* [2] propose a database column embedding using Bi-LSTM for drug-drug interaction (DDI) prediction. Cappuzzo *et al.* [10] propose data integration using graph-based representation. Günther *et al.* [17] proposes enriching database queries using pre-trained *word2vec* model.

Online index tuning: The problem of online indexing has been studied widely. The preliminary work was done by Hammer *et al.* [19] in the 1970s, where authors use heuristics for index selection on a single table. Frank *et al.* [16] proposed an online tuning method for a single index selection problem using workload statistics for change detection and using heuristics for index selection. Kołaczkowski *et al.* [25] searches for indexes in a query execution plan space and select them using evolution strategy. A solution using heuristics may fail in scalability, specifically in challenges (C4). Bruno *et al.* [8] design a hardcoded rule approach for index selection. Their approach is fast and scalable, but it lacks noise resilience (C1). For a small DBMS application such an approach may work, but it can worsen the performance on a larger system. Schnaitter *et al.* [50] proposed a very effective online indexing solution that monitors the incoming queries and minimizes the overhead cost. Their index selection uses heuristics. For real-time online index selection it is slow (C4) and susceptible to noise (C1). Sadri *et al.* [49] proposed an index tuning algorithm using deep reinforcement learning. As described by the authors, their approach takes a long time for index selection. Thereby not being able to handle the challenge (C4). The previous approaches does not overcome all challenges. They mostly fail in noise resilience (C1), and scalability (C4). Our proposed approach overcomes all the challenges (C1)-(C4). We discuss them in detail in Section 4.3.5.

3 PROBLEM FORMULATION

The index recommendation problem is to select a set of indexes that minimizes the time to evaluate a workload and the amount of storage needed. A set of indexes that minimizes the workload cost with a storage size constraint is a *constrained combinatorial optimization* problem.

A workload W is a set of SQL queries Q_1, Q_2,.., Q_m. An index configuration I is a set of indexes. We calculate the cost of workload evaluation on database D using the cost of the evaluation of each individual query given by $Cost(Q_j, I, D)$. The cost of a workload can be described as follows.

$$Cost(W, I, D) = \sum_{j=1}^{m} Cost(Q_j, I, D)$$

Note that the workload cost does not weight queries differently, though we could trivially include such weights by replicating individual queries in a workload. The index selection problem is to find a set of indexes $I_{optimal}$ that minimizes the total cost of a workload, $Cost(W, I, D)$, and has a storage cost of at most C.

$$I_{optimal} = \min_{S(I^*) \leq C} Cost(W, I^*, D)$$

In this equation $S(I^*)$ is the total storage space cost of the set of indexes I^*. The space constraint is important because index creation should be *parsimonious*. Creating more indexes than necessary wastes space and slows data modification.

The online index selection problem can be formulated as follows. Let a workload stream W^* be a sequence of workloads $[W_1, W_2, .. , W_k]$. Let $Diff(W_1, W_2, .., W_k)$ be a metric that quantifies the difference between workloads in the stream. Then *online Index Selection* can be defined as:

$$reindex_{config}(I^{**}, D) = \min_{Diff(W^*) \geq \lambda} Cost(W^*, I, D) \quad (1)$$

where, λ is a measure of the sensitivity to re-configure the indexes based on the workload stream difference. λ is described in detail in Section 4.2.1.

4 INDEXER++

Indexer++ has two phases. The first phase identifies workload trends. It uses query embedding to model a workload. The second phase tunes the set of indexes using deep reinforcement learning. This section describes the two phases.

4.1 Workload Embedding

Modern data-based systems are designed to support hundreds of thousands of users using cloud computing. The continuous tuning of such a database management system has become challenging. The performance of the database management system is highly dependent on its configuration, which is chosen based on workload analytics. A database management system (DBMS) must be able to adjust to evolving workloads in order to provide optimal performance. There is a growing need for a workload-aware self-tuning database management systems. A successful workload-aware DBMS would be able to tune itself by adapting to workload patterns [45].

There are several challenges in building such a DBMS [59], such as: **(i) Heterogeneous:** a DBMS consists of various types of workload such as analytical, data manipulation and data definition. They have different functionality, and execution cost. The heuristics-based pattern extraction is difficult for such workloads. It requires a solution that is able to learn and adapt to heterogeneous workloads. **(ii) Scale and Volume:** with the advent of cloud computing there has been tremendous growth in data based applications. The applications are frequently created and are utilized by existing and new users. This adds large number of queries with changing workload patterns. Such rapid change in workload trends requires an online process of understanding queries. **(iii) Representation:** a workload-aware DBMS requires extracting patterns in real-time. Understanding syntactic and semantic meaning of a workload requires it to be represented in a standard form. It requires a pattern extraction approach that must be able to learn the relationship among workload and withstand the workload volume.

A workload-aware DBMS requires a deeper understanding of input queries. A SQL query workload consists of patterns related to users and applications. Extracting such patterns may have several applications such as workload compression and index recommendation [23]. Due to increase in volume of data and data-based applications, there is a need for a generalized framework to extract such patterns. This problem is similar to several NLP based applications, where input text is represented in a standard form for analysis. To learn syntactic and semantic meaning of words, *Representation Learning* [4] is heavily used in NLP. The main idea of text embedding using *Representation Learning* is to fix a d-dimension vector for each word in a corpus of text [6]. A word is represented in high-dimensional vector space as contrary to a sparse, discrete vector using one-hot encoding. This representation encodes the meaning of a word. They have shown to capture latent features

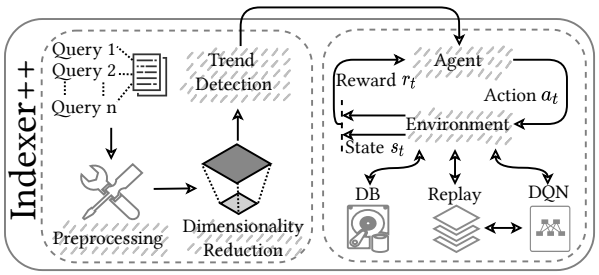

Figure 1: Indexer++ Framework

of a word, in such a way that relationship between words can be expressed using arithmetic operations *e.g., king to man is what to woman?* Answer: queen [34]. These techniques can be extended to the SQL query embedding.

There has been previous work related to workload representation. Typically, there has been two proposed ways to represent a workload **(i)** using statistics of database by executing workload [58] **(ii)** training deep learning models on large corpus and performing inference [23]. Both of these approaches have drawbacks, former approach rely on database statistics, to extract such information the workload needs to be executed which may interrupt ongoing processes and later approach, requires training during the runtime, which is time and computationally expensive. A much realistic approach could be performed using *universal embedding*. It is a form of *transfer learning*, in which model is pre-trained on a very large corpus of text. There are few advantages of using *transfer learning* over existing approaches. Firstly, the inference time [1] has no slack which makes it a feasible solution for online tuning (overcoming challenge (ii) **Scale and Volume**). Secondly, it does not collect any info from database thereby not interrupting any processes or application running. In our experiments, we found *universal embedding* to be effectively able to **Represent** workload and have the ability to identify **Heterogeneous** workloads. We experiment with several pre-trained transformer based BERT models and variants as shown in Section 5.2. Our approach is able to overcome all three challenges **Heterogeneous**, **Scale and Volume**, and **Representation** discussed earlier.

4.2 Detecting Change in Workload Trend

So far, we have discussed about transforming a workload to a vector representation *i.e., Workload Embedding*. There are several applications of a such embedding for example, workload summarization, cost estimation and query classification. In this section, we will explore *Workload Trend Detection* an another application of workload embedding that has not gain much popularity.

Lets presume the *App-X* framework for trading on the stock exchange, where users can check for a ticker (abbreviation of stock), conduct chart analysis, purchase and sell stocks. On a trading day, value-based stock searches could be strong at the opening bell of the market and, on the same day, technology-based stock searches could gain traction near the closing bell. These systems suffer from a massive shift in search patterns. We may configure a DBMS to manage such change in search queries, by having the ability to

[1] time required to convert a word to vector representation using pre-trained model

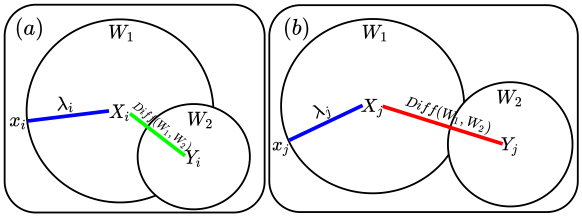

Figure 2: Configuration of trigger parameters λ and $Diff(W_1, W_2)$

detect such patterns. This, in essence, would boost performance and user experience with the *App-X*. With the ease in availability of internet and social media, applications with swiftly moving patterns are thriving. For such applications, a database must be able to adapt to the changing environment for better performance. This requires a DBMS to be able to autonomously detect these rapid change of trends and perform configuration adjustments. For such a task, we design and develop a *Workload Trend Detection* based on *Workload Embedding*.

The objective of *Workload Pattern Detection* is to examine the variation between workloads and trigger whether the DBMS needs to be reconfigured. This task can be outlined as follows: Given two workloads (W_1, W_2) obtained from a DBMS at two separate periods. Given a metric function $Diff(W_1, W_2)$, that performs workload embedding, analyzes and quantifies the difference between workloads. This metric is used to define similarity or dis-similarity between workloads. Next, with the help of a sensitivity threshold λ, we trigger a DBMS reconfiguration when $Diff(W_1, W_2) \geq \lambda$.

To achieve *Workload Trend Detection* task, we start with workload embedding. We then perform dimensionality reduction using *t-distributed stochastic neighbor embedding* (t-SNE), a non-linear dimensionality reduction technique. The feature space of t-SNE is found by searching for a low-dimensional projection of data which is the closest equivalent to the real data using a stochastic neighbor embedding. The neighbor embedding are constructed in two stages: (i) it constructs probability distribution such that similar objects are assigned high portability, where as dis-similar as lower. (ii) it constructs a similar distribution in lower dimension and minimizes the entropy using KL Divergence of the two distribution. This lower dimensional data is widely used to produce scatter plots for visual data analysis. We perform such an analysis on a real life `IMDB` dataset (dataset description in Section 5.1), as shown in Fig 3. We perform experiments on several pre-trained models, dimensionality reduction techniques and clustering algorithms, described in detail in Section 5.2. We identify t-SNE, ROBERTA, and k-medoids to be the best dimensionality reduction, pretrained NLP model and clustering algorithm respectively.

Next, we assume there are two cluster of patterns in a workload: historic (W_1) and upcoming/future workload pattern (W_2). With the assumption in mind we perform clustering on the dataset. Specifically, we perform *k-medoids* clustering on the reduced dataset (using t-SNE). We use *k-medoids* over *k-means* and *k-means++* due to interpretability of the selected centroids. The *k-means* and *k-means++* returns the centroid that are not an actual data points, instead they are an average of data points. However, selected centroid using *k-medoids* is an actual data from the dataset. The selected centroid by *k-medoids* can be traced back to the actual query. It helps in interpretation and visualization of our framework. The dimension reduction is a standard pre-processing technique for clustering algorithms, as it decreases the noise in the data [22]. This preprocessing improves the overall quality of the cluster.

Next, we analyze the distance between historic (W_1) and future workload (W_2). We select the centroid of both workloads. We then calculate euclidean distance between centroids $Diff(W_1, W_2)$. A DMBS reconfiguration is triggered based on the value of λ. The value of λ (a sensitivity parameter) varies on different size of systems. We explain in detail the values of λ and $Diff(W_1, W_2)$ in Section 4.2.1. After reconfiguration, we merge future workload to historic workload. This is an iterative process. The workload trend detection requires minimum of n queries in current workload, this help in mitigating the affect of outliers. In general, the value of $n \geq 32$, it is same as the batch size for DQN algorithm.

4.2.1 ***Value of λ and $Diff(W_1, W_2)$:*** A reconfiguration of database is required with change in workload trends. To observe the change in workload trend, we assume there are two workloads (i) existing, and (ii) future workload. A workload has a span when represented in a high-dimensional vector space. We assume the span of a workload to be a sphere. The sphere is centered at computed cluster centroid. The radius of sphere is calculated using farthermost point from centroid of the workload. The radius of sphere is termed as λ in our framework. The value of λ represents the spread of workload in vector space. We compute the centroid of future workload and represent it as an another sphere. We define the distance between both sphere using the euclidean distance between respective centroids, termed as $Diff(W_1, W_2)$. The value of $Diff(W_1, W_2) \leq \lambda$, indicates that the future workload is similar to the existing workload. since, the centroid of future workload lies in the span of existing workload. However, if $Diff(W_1, W_2) \geq \lambda$, it suggests that the future workload is different. The DBMS will requires a reconfiguration and recomputing of centroid.

We describe our idea using visual representation in Fig 2. We have two scenarios (*a*) and (*b*). In the figure, X and Y represents centroid of existing (W_1) and future workloads (W_2), respectively. The value of λ represents the distance between centroid and furthest point and the $Diff(W_1, W_2)$ quantifies the distance between centroids, X and Y. In scenarios (*a*,) the reconfiguration is not requires as $Diff(W_1, W_2) \leq \lambda$, instead the future workload is merged in the existing workload. In the second case (*b*), the reconfiguration is triggered as $Diff(W_1, W_2) \geq \lambda$ and new indexes are selected.

Trend Detection on `IMDB`: To visually describe and evaluate our workload trend detection approach we perform an experiment on `IMDB` dataset. The `IMDB` queries are split into three groups, namely *Group 1*, *Group 2*, and *Group 3*, where each group represents a set of similar queries. There are same number of queries in each group. The *Group 1* is assumed as historical workload and *Group 2* and *Group 3* are introduced later as future workloads. The SQL queries are pre-processed by removing SQL keywords and an inference in performed using pretrained ROBERTA. The output of inference returns a *high*-dimensional vector. The output is reduced to two-dimension using t-SNE. A visual of the *Group 1* in reduced dimension is shown in top-left corner of Fig 3. The k-medoids clustering on reduced dataset returns the centroid shown in red circle

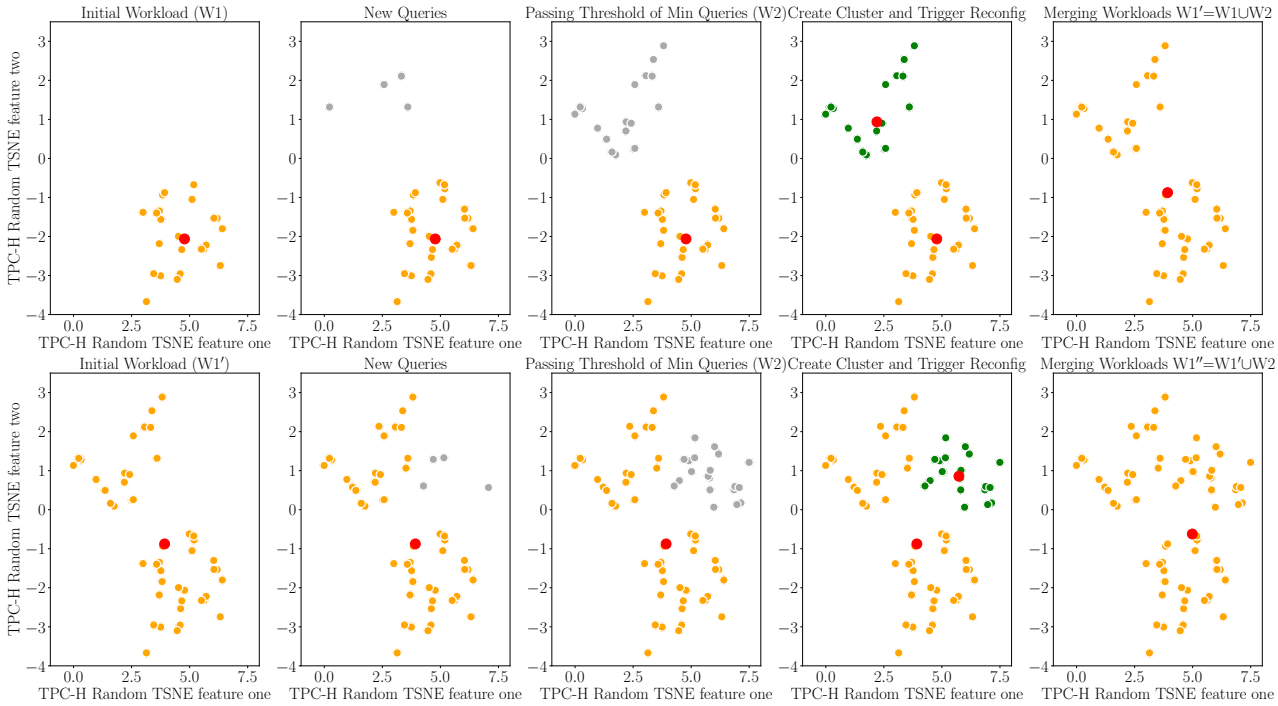

Figure 3: Workload Trend Detection on three groups of TPC-H Random dataset where red circle indicates the centroid of the cluster. (figure is better viewed in color)

in the figure. A few queries are introduced from *Group 2*, the visual next to top left corner shows a clear distinction between *Group 1* (in orange) and few queries from *Group 2* (in gray). The next visual consist of all the queries from *Group 1* and *Group 2*. After surpassing minimum queries n required for a future workload, clustering is performed (in green) and a data point in red circle shows the centroid. The clusters are merged based on the $Diff(Current, Future) \geq \lambda$ and the centroid is recomputed. At this time a reconfiguration is also triggered to add new indexes for the *Group 2* workload. A complete description and analysis of values of $Diff(W_1, W_2)$ and λ is described in a later Section 4.2.1. In a similar way, in the second row of the figure *Group 3* is introduced. The centroids are recomputed, and workload is merged to the cluster. A set of indexes for *Group 3* are also introduced.

4.3 Online Index Recommendation

Section 3 describes the online index selection problem. We observe the indexes in a database are intertwined in the sense that query optimization may take advantage of the presence of several indexes in combination to optimize a query, an optimization opportunity that may be missed if one of the indexes were absent. This means that the online index problem can be modeled as a sequential decision making process where we can optimize the performance of DBMS at every episode/step. Recently, *Reinforcement Learning* (RL) has become a popular technique to optimize sequential decision making problems, also known as *Markovian Decision Processes* (MDPs). RL has been used to approximate solutions to various applications of MDPs such as routing vehicles [43], discovery of directed acyclic graphs [60], and the traveling salesman problem [41]. In contrast to traditional *Machine Learning*, where training requires a labeled dataset, an RL agent interacts with an environment and learns from *experiences*. In a typical RL setup, an agent interacts with an environment to learn an optimal control policy [55]. At a given time t with state s_t, an agent selects an action $a_t \in A$, following a policy $\pi(s_{t+1}|s_t, a_t)$ proceeding to next state s_{t+1}. It receives a scalar rewards r_t during state transition.

We formulate our online index selection problem as a Markovian process. We extract relevant information from a database to define a *state* and design a pipeline where at time t we have an $action_t$ to $state_t$ mapping. In this scenario, we have a deterministic state and action. The goal of MDP is to reach the final state maximizing cumulative rewards $C(r)$ and identifying a *policy* $\pi(s, a)$: action-state mapping which selects the optimal action $a \in A$ at given state $s \in S$.

4.3.1 ***Online DQN Algorithm.*** In a popular offline RL based algorithm Deep Q-Networks (DQN), policy $\pi(s, a)$ and values $q(s, a)$ are represented using multi-layer neural networks. The neural networks applies high-dimensional input data representation, generalizing similar experiences and unseen states. The hyper-parameters of neural networks are trained by gradient descent by minimizing a loss function. We use the mean squared error as loss function shown below:

$$Loss(\theta) = \mathbb{E}_{\pi}\left[\frac{1}{2}\overbrace{(Bellman}^{Target} - \overbrace{Q(s, a; \theta))^2}^{Estimate}\right]$$

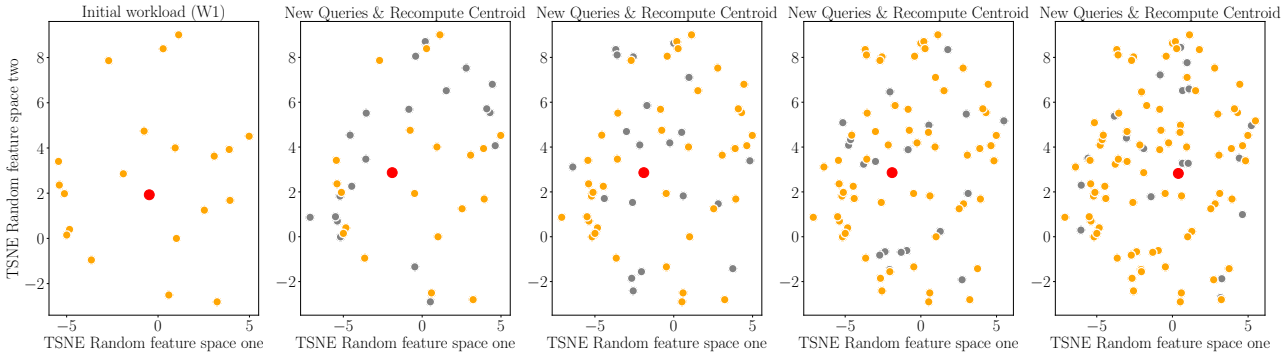

Figure 4: Workload Trend Detection on TPC-H dataset. It is a scenario when no re-configurations of indexes is trigger. The new queries are similar to existing workload. The red circle indicates the centroid of the cluster. The existing workload is displayed in orange and new queries are in gray. (figure is better viewed in color)

where θ is a parameter for a nonlinear function, in our case a neural network. Expanding the above equation using Bellman Optimality equation [56] we get:

$$Loss(\theta) = \mathbb{E}_{\pi}\left[\frac{1}{2}\left(\overbrace{(R(s,a)}^{Reward} + \overbrace{\beta \max_{a' \in A} Q(s',a';\theta))}^{Future\ Reward} - \overbrace{Q(s,a;\theta)}^{Estimate}\right)^2\right]$$

where, β is the discount rate. In above equation, we approach to learn weights of $Q(s,a;\theta)$. We use gradient descent optimization algorithm to learn the parameters and a gradient update w.r.t $Q(s,a;\theta)$ can be perform using below:

$$Q(s,a;\theta) = Q(s,a;\theta) - \alpha \frac{\partial}{\partial Q(s,a;\theta)} Loss(\theta)$$

replacing $Loss(\theta)$ and taking a partial derivative yields:

$$Q(s,a;\theta) = Q(s,a;\theta) + \alpha\left(\underbrace{\overbrace{R(s,a) + \beta \max_{a' \in A} Q(s',a';\theta)}^{TD\ Target} - \overbrace{Q(s,a;\theta)}^{Estimate}}_{Temporal\ Difference\ (TD)\ Error}\right)$$

Rearranging the above equation gives:

$$Q(s,a;\theta) = (1-\alpha)\ Q(s,a;\theta) + \alpha\left(R(s,a) + \beta \max_{a' \in A} Q(s',a';\theta)\right)$$

We use above equation to generalize the approximation of the Q-value function. A neural network training assumes that input data are independent and sampled from similar distributions. A neural network will overfit/underfit if such assumptions are not satisfied. For Reinforcement Learning, we can observe that the target Q value depends on itself, making training on a neural network difficult since it would chase a non-stationary target. To solve this problem, we implement a target network θ', which stays stationary for a certain period and later synchronizes with θ. The above equation can be re-written using target network θ' as follows:

$$Q(s,a;\theta) = (1-\alpha)Q(s,a;\theta) + \alpha\underbrace{\left(R(s,a) + \beta \max_{a' \in A} Q(s',a';\theta')\right)}_{TD\ Target}$$

4.3.2 ***DBMS representation:*** A *state* of a DQN agent represents an environment. A state representation should be able to describe the environment to the finest detail. It is a crucial component for learning an optimal policy. We use index configuration of a DBMS as state representation. Specifically, index configuration is represented in a one-dimensional vector. In the vector, a binary value represents presence and absence of an index. The length of vector is defined by the value of all possible index configuration $(2^{N_{columns}} - 1)$. The *action* space represents all possible index configuration actions that can be performed and is also the same size as state space. A *reward function* is an another crucial component of the learning process of DQN agent. It defines a cost function for the agent, with a goal to maximize. We define our reward function as below:

$$rewards_{size} = \begin{pmatrix} 1 & \text{index size} < \text{max allowed size} \\ -1 & \text{otherwise} \end{pmatrix}$$

$$r_t = \max\left(\frac{\text{index cost}}{\text{all index cost}} - 1, 0\right) + rewards_{size} \quad (2)$$

where, numerator is the workload cost with selected index and denominator is workload cost with all index configuration. Our reward function is designed such that it will have minimum value of -1 and maximum of 10-20, such smaller range of rewards suppresses the noise while training Neural Network. We also introduce the reward for disk size constraint, where total size of selected indexes is upper bounded with *max_allowed_size e.g.,*. 10MB. Our reward function is generic and can be modified for any other constraints.

4.3.3 ***Priority Experience Replay:*** A Reinforcement Learning algorithm updates while interacting with the environment. In such a situation, the algorithm tends to forget experiences after a few epochs of training. To solve this problem, we store a buffer called *Experience Replay*, which is used to sample from previous experiences. It also breaks the temporal dependency otherwise found in regular updates while interacting with environment. *Experience Replay* can lead to slow convergence because data is sampled uniformly. A previous experience with a large estimated error may or may not be in sampled data, making the convergence slower. To solve this problem, we use *Priority Experience Replay*, where instead of sampling uniformly, samples are giving importance. We use *Temporal Difference Error* to prioritize experiences:

Algorithm 1 Online DQN Algorithm with Priority Experience Replay and Sweeping

Initialize batch size N_s ▷ 32
Initialize number of Iterations I_s ▷ 1000
Initialize length of a episode L_s ▷ 3-6
Initialize update target network frequency F_s
Initialize priority scale η ▷ 1.0
Initialize priority constant ϵ ▷ 0.1
Initialize network parameter θ
Initialize target network parameter θ'
Initialize learning rate α ▷ 0.001
Initialize discount factor β ▷ 0.97
for each $l \in L_s$ **do** ▷ number of episodes
  Collect experiences (s, a, r, s', p) ▷ until minimum N_s size
  for each $i \in I_s$ **do**
    Sample N_s prioritized samples from buffer
    for each $n \in N_s$ **do**
      $y_i = r_i + \beta \max_{a'_i \in A} Q_{\theta'}(s'_i, a'_i)$
      if done == $True$ **then** ▷ check for terminal state
        $\delta_i = |y_i - Q_\theta(s_i, a_i)|$ ▷ calculate TD Error
      end if
    end for
    $L(\theta) = \frac{1}{N_s} \sum_i (y_i - Q_\theta(s_i, a_i))^2$ ▷ calculate loss MSE
    $\theta = \theta - \alpha \nabla_\theta L(\theta)$ ▷ update network parameters
    $p_i = \frac{(|\delta_i|+\epsilon)^\eta}{\sum_j (|\delta_j|+\epsilon)^\eta}$ ▷ calculate and update samples priority
  end for
  if `reconfig` == $True$ **then** ▷ workload trend detection trigger
    `empty replay`
  end if
  if l mod F_s **then**
    $\theta' = \theta$ ▷ update target network
  end if
end for

$$\delta = R(s, a) + \beta \max_{a' \in A} Q(s', a'; \theta') - Q(s, a; \theta) \quad (3)$$

Using *Priority Experience Replay* we ensure that our network learns from experiences which it found hard to generalize, and it also helps in faster convergence [20].

4.3.4 ***Priority Experience Sweeping:*** By nature, DQN algorithm is an online learning process. It interacts with an environment and by maximizing rewards learns the optimal behaviour. Similarly, in index tuning process by reducing the query cost, DQN agent learns to recommend suitable indexes for better query performance. However, when the trend in workload changes, the optimal policy of the environment will change and that may require retraining of the DQN agent. The retraining process is time consuming and if the process takes longer it will reduce the utility of have online index tuning. To avoid retraining and to handle this scenario we introduce *Priority Experience Sweeping*. The motivation for *Priority Experience Sweeping* is that DQN agent has already learned optimal policy for the environment. The state, action of environment does not change, instead there is a slight change in the behavior. Rather than retraining from scratch, *how can we utilize previous learned behavior?*

Example: To give the context, we describe *Priority Experience Sweeping* using an example of a robot navigation in a warehouse.

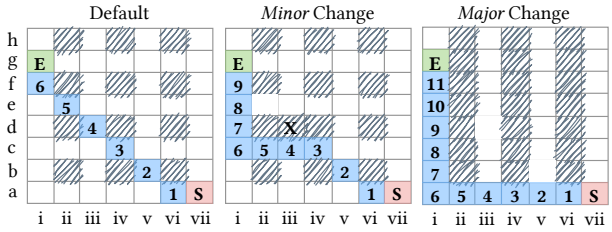

Figure 5: A robot navigation in a warehouse from a start point (in red) to end point (in green), blue boxes show an optimal route and gray boxes are obstacles (better view in color)

Given a warehouse where a robot is required to navigate from a start point to an end point avoiding obstacles as shown in Fig 5 (*Default*). A DQN agent can learn this behavior by trail and error and can discover the optimal path for navigation as shown in the same figure with blue boxes. Consider a slight change in the environment by adding a new obstacle as shown by **X** (location: d-iii) in *Minor* Change section of Fig 5. With this minor change in the environment, the optimal path has changed. However, it has a few similarity to the previously learned path from *Default* environment. The location of end point, few initial steps, and direction to navigation learnt from *Default* environment are the same. Instead of re-training a DQN agent, the previously learned behaviour from *Default* environment can be utilized and in addition to that minor changes can be learned. A DQN agent stores its previous experiences in replay buffer. The neural network samples data from this buffer for training. When a change is observed in the environment the previous experiences become irrelevant. The policy to interact in the newer environment requires experiences from the newer environment. Using this idea, when a change in the DBMS workload pattern is triggered we remove all experiences from replay buffer. The replay buffer is reloaded with the newer experiences. This gives the agent opportunity to sample from newer experiences and learn the optimal policy for the newer environment.

In robot navigation example, the policy learned from *Default* environment when executed in *Minor* environment, it will suggest it to follow the previously learned path and robot will hit the obstacle **X**. We load these failure experiences in the buffer. In order to maximize rewards (cost function), DQN agent will explore alternatives. The agent has a prior knowledge from *Default* environment and in a few iteration of training the agent is able to navigate the robot to the an alternative path, without retraining from scratch. If there are *Major* changes to the environment such an approach still (end point, direction of navigation and initial step remains the same) works but this may take longer to adjust to the newer environment.

In a similarly way, patterns in DBMS workload could be a *Minor* or *Major* change. With the usage of *Priority Experience Sweeping* our framework can ajust to the changes without retraining from scratch. After a workload trend detection trigger a signal for change in the environment we remove existing experiences from the priority experience replay and reload it with newer environment experiences. We train for few iterations and observe that agent was able to learn the behavior of newer environment without retraining. This

Algorithm 2 Random Query Generator

```
Initialize maximum number of columns in a query C        ▷ 1-4
Initialize number of queries Q                           ▷ 100
for each q ∈ Q do                                        ▷ number of queries
    for each c ∈ C do                                    ▷ number of columns
        Randomly extract a distinct value
        Randomly select operator [>, <, =, =>, <=]
        Randomly select predicates [and, or]
        Append q
    end for
end for
```

process makes our framework a generic tool for online configuration tuning. The complete DQN algorithm with *Priority Experience Replay* and *Priority Experience Sweeping* is shown in Algorithm 1.

4.3.5 ***Overcoming Challenges:*** In Section ??, we discussed about several challenges in achieving an online index tuning. We design and build our framework to handle all the challenges. We describe solutions to each of the challenges below:

(C1) **Noise Resilience**: We threshold a minimum number of query in a workload for trend detection. This ensures a few outliers do not trigger the reconfiguration. We also perform clustering on the dataset, that helps to reduce affect of outliers. We also use a threshold λ as sensitivity parameter towards trend detection trigger. Overall, these thresholds help mitigate noise in the workload.

(C2) **Overhead Cost**: In our approach for learning the cost of an index we rely on in memory hypothetical indexes[2], contrary to other approaches of creating an actual index. It works independent of any other DBMS application running. This ensures the overhead cost of index tuning do not affect DBMS performance and our framework can be used concurrently to active DBMS nodes.

(C3) **Trend Detection Time**: We perform workload embedding and analytics on pretrained BERT models. In our approach, we do not train them and only use inference of the models. It requires a very short duration of time for inference from pretrained models.

(C4) **Response Time**: We use pretrained NLP model for workload analytics and DQN for index tuning. We also eliminate re-training of DQN algorithm using *Priority Experience Sweeping*. This reduces the overall cost of response time. Our framework can analyze workload and recommend indexes in a short period of time (15-20min).

5 EXPERIMENTS

5.1 Dataset Description

(1) **IMDB**: It is a real life dataset about movies and related facts about actors, directors etc [32]. The dataset is highly correlated and it relates with a real-world scenario. It consists of 21 tables and a set of 33 queries. The each set consists of 3-4 queries, and in total consists of 109 queries. In our experiments, we divide 33 set of queries in 3 groups. The set 1-10 creates the `Group 1`, set 11-20 is `Group 2` and set 21-33 is `Group 3`. The purpose of forming groups is to analyze the performance of our framework with changing workload trend. Each group is used as a workload, we start with `Group 1` as current workload and periodically introduced `Group 2` and 3. The idea is to analyze and observe trigger detection in changing workload environment.

(2) **TPC-H**: A standard benchmark dataset published by Transaction Processing Performance Council (TPC). The queries and data in the dataset are chosen to reflect industry-wide relevance. It illustrates systems that examine large volume of data and executing complex queries. The `TCP-H` consists of a sub-program Q-gen to generate queries. The Q-gen program consists of 22 query templates. The query templates are not enough to define the diversity of a real-time workload. Having that in mind, we build a random query generator to perform experiments. Our query generator is a generic code base which can generate random queries from any database. Our approach is described in Algo 2. A few of a sample queries are shown below:

```
SELECT COUNT(*) FROM LINEITEM WHERE L_PARTKEY = 30217
SELECT COUNT(*) FROM LINEITEM WHERE L_ORDERKEY = 7919908
SELECT COUNT(*) FROM LINEITEM WHERE L_SUPPKEY = 14816
SELECT COUNT(*) FROM LINEITEM WHERE L_SHIPDATE > 1997-05-23
```

In this paper, we refer to randomly generated queries as `TPC-H Random`, template queries as `TPC-H Template` and 33 set of queries from IMDB as `IMDB Dataset`.

5.2 Experimental Setup and Results

In this section we describe our experiments, experimental setups, and results. All experiments in our experiments were performed on a computer with Intel i7 5820k, Nvidia 1080ti, 32GB of RAM running Ubuntu 18.04 OS. We use Python 3.7 and libraries (powa, gym, psycopg2, sklearn, TensorFlow, Keras) to write and train our framework. The DQN was trained on Nvidia 1080ti (3584 CUDA cores and 11GB DDR5 RAM) with CUDA and cuDNN configured for performance enhancement.

5.2.1 ***Experiment Objective 1:*** *What is the most effective combination of Workload Representation, Clustering, and Dimensionality Reduction algorithms for SQL queries?*

To the best of our knowledge, there has been no such previous study/experiment to analyze the best available algorithm for these tasks on SQL queries. We design an experiment by selecting few popular algorithms for each task. For better evaluation, we perform our experiment on two different dataset `IMDB` and `TCPH Random`. The selected algorithms are briefly described below:

Pretrained NLP Models:

(1) **BERT** [14]: It uses a combination of Masked Lanaguage Model and next sentence prediction in a bidirectional transformer which is pretrained on a large english lanagueg corpus of Toronto Book Corpus and Wikipedia.

(2) **ALBERT** [30]: It is bidirectional transformer based model similar to BERT. It uses parameter-reduction techniques for lower memory and higher speed of training. It is a miniature version of BERT.

[2]https://github.com/HypoPG

Table 1: IMDB Dataset Queries

	BERT			ALBERT			RoBERTa			XLM			Transformer-T5		
	K-Mn	K++	K-Md	K-Mn	K++	K-Md	K-Mn	K-M	K-Md	K-Mn	K++	K-Md	K-Mn	K++	K-Md
UMAP	0.759	0.759	0.776	0.732	0.732	0.732	0.786	0.786	0.768	0.768	0.768	0.750	0.750	0.750	0.759
PCA	0.750	0.750	0.750	0.723	0.723	0.723	0.750	0.750	0.750	0.741	0.741	0.741	0.732	0.732	0.732
T-SNE	0.759	0.759	0.767	0.714	0.714	0.741	0.741	0.759	0.786	0.768	0.768	0.776	0.741	0.741	0.741

Table 2: TPC-H Random Dataset Queries

	BERT			ALBERT			RoBERTa			XLM			Transformer-T5		
	K-Mn	K++	K-Md	K-Mn	K++	K-Md	K-Mn	K++	K-Md	K-Mn	K++	K-Md	K-Mn	K++	K-Md
UMAP	0.849	0.849	0.836	0.685	0.685	0.685	0.973	0.973	0.945	0.753	0.753	0.753	0.822	0.822	0.822
PCA	0.671	0.671	0.657	0.493	0.671	0.753	0.904	0.904	0.890	0.986	0.986	1.00	0.822	0.822	0.836
T-SNE	0.808	0.808	0.877	0.671	0.671	0.671	0.959	0.959	0.904	0.986	0.986	0.753	0.932	0.932	0.932

Table 3: IMDB Queries with no SQL Keyword

	BERT			ALBERT			RoBERTa			XLM			Transformer-T5		
	K-Mn	K++	K-Md	K-Mn	K++	K-Md	K-Mn	K-M	K-Md	K-Mn	K++	K-Md	K-Mn	K++	K-Md
UMAP	0.660	0.660	0.652	0.664	0.664	0.664	0.768	0.768	0.776	0.768	0.768	0.786	0.794	0.794	0.794
PCA	0.661	0.670	0.642	0.664	0.664	0.664	0.750	0.750	0.750	0.741	0.741	0.750	0.768	0.768	0.768
T-SNE	0.688	0.688	0.679	0.664	0.664	0.664	0.750	0.750	0.812	0.786	0.768	0.759	0.786	0.786	0.804

Table 4: TPC-H Random Dataset with no SQL Keyword

	BERT			ALBERT			RoBERTa			XLM			Transformer-T5		
	K-Mn	K++	K-Md	K-Mn	K++	K-Md	K-Mn	K++	K-Md	K-Mn	K++	K-Md	K-Mn	K++	K-Md
UMAP	0.808	0.808	0.794	0.671	0.671	0.740	0.835	0.835	0.835	0.712	0.712	0.712	0.726	0.726	0.698
PCA	0.575	0.575	0.575	0.699	0.699	0.699	0.917	0.917	0.917	0.657	0.657	0.712	0.534	0.534	0.534
T-SNE	0.589	0.589	0.603	0.767	0.767	0.767	0.849	0.849	0.834	0.712	0.712	0.726	0.671	0.671	0.656

*K-Mn: K-Means, *K++: K-Means++, *K-Md: K-Medoids

(3) **RoBERTa** [38]: It is also a variant of BERT model. It modifies key hyper parameters and are trained with large batch size and learning rate. It is a very fined tuned version of BERT.
(4) **XLM** [28]: It is a cross-lingual model using one of the casual language model, masked language model and translation language model as objectives for various tasks.
(5) **Transformer-T5** [12]: It is a uni-directional transformer with relative position embeddings. It uses hidden states of previous layers (residual) for longer memory. It does not have a length of sequence (input length) limit.

Dimensionality Reduction:

(1) **Principal Component Analysis** (PCA): It is a linear, matrix factorization based dimensionality reduction method. It works by computing a hyperplane closest to the dataset. While maintaining the variation of dataset it projects the data on the computed hyperplane. The output of PCA are interpretable.
(2) **t-Stochastic Neighborhood Embedding** (t-SNE): It is a non-linear and graph based method. It works by defining probability distribution of dataset in high and lower dimension. Using KL divergence it minimizes both distributions. It does not preserve the global structure only local.
(3) **Uniform Manifold Approximation and Projection** (UMAP): It is a non-linear and topological structure based method. It uses exponential probability distribution in high dimensions instead of euclidean distance by t-SNE. It preserves both local and global structure of the dataset. It is well suited for cluster analysis.

In our experiment, we select K-Means, K-Means++ and K-Medoids as clustering algorithms. We start with workload representation using pretrained NLP models. The NLP models are trained on word tokens and adopt a set of rules for input text processing. Our dataset must follow the same token format. For initial preprocessing, we use the learned tokenizer from pretrained models. Tokenization is the method of breaking down a text into words/sub-words. The subwords are converted to IDs using a *Vocab* that is unique to the pretrained models and was learned from a large corpus. Byte-Pair Encoding (BPE), WordPiece, and SentencePiece are three popular tokenization methods. The tokenization algorithm used for model training differs, and we use the tokenizer that was used originally to train the model. We perform tokenization of workload that returns

Table 5: Average accuracy from both datasets

With SQL Keyword Vector Representation		No SQL Keyword Vector Representation	
BERT	0.76987524	BERT	0.6621004566
ALBERT	0.6956675364	ALBERT	0.5731065045
RoBERTa	**0.8492199391**	RoBERTa	**0.8149733638**
XLM	0.5686494347	XLM	0.4783172972
T5	0.8010165253	T5	0.7119958143
With SQL Keyword Clustering		**With SQL Keyword Dimentionality Reduction**	
K-Means	0.7843118069	UMAP	0.7860159817
K-Means++	**0.7908431181**	PCA	0.7719259622
K-Medoids	0.7869903783	TSNE	**0.8042033594**
No SQL Keyword Clustering		**No SQL Keyword Dimentionality Reduction**	
K-Means	0.7212273566	UMAP	**0.7423297701**
K-Means++	0.7209297375	PCA	0.6985186155
K-Medoids	**0.7257527969**	TSNE	0.7270615053

a high-dimensional tensor representation. We apply dimensionality reduction and perform clustering on the representation. Both of the datasets are pre-labeled with respective clusters. The labeling on the `IMDB dataset` is based on the groups and for `TPC-H Random` is based on the columns used in the queries. For `TPC-H Random` dataset, we randomly select three columns and generate 25 queries each with single columns, in total of 75 queries. We measure the accuracy of the computed clusters (sorted w.r.t IDs) using below:

$$accuracy(y', y) = \frac{1}{n}\frac{1}{m}\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(y'_{ij} == y_{ij}) \begin{cases} 1, & \text{if, } equal \\ 0, & otherwise \end{cases}$$

where, n is the number of clusters and m is the number of data points in the cluster. We perform our experiment on both datasets by keeping and removing SQL keywords such as, SELECT, COUNT, etc. The results are outlined in Tables [1, 2, 3, 4, 5].

Specifically, a pre-trained model is selected *e.g.,* BERT on `IMDB dataset` with SQL keywords. An inference is perfomed using BERT and dataset is reduced using UMAP, t-SNE, PCA. A clustering is performed using K-Means, K-Means++ and K-Medoids and quality of the cluster is evaluated in terms of accuracy. Iteratively this experiment is performed on other NLP models ALBERT, RoBERTa, XLM, Transformer-T5 and the accuracies are reported in Table 1. We repeat this experiment on `TPC-H Dataset`, results are shown in Table 2 and also by removing SQL keywords in both datasets, results are in Table 3, 4. To identify the most effective method we calculate the average of cluster accuracy for all experiments and report them in Table 5. We observe,

- Cluster detection accuracy with SQL keywords is 3.4% better than without SQL keywords.
- The average accuracy of RoBERTa pretrained model ourperformed other models on both with and without keywords.
- The K-Means++ performs best with keywords and K-Medoids is best without keywords. Taking the average of both (with and without keyword) K-Medoid performs better than K-Means++.
- Dimensional reduction algorithm t-SNE and UMAP performs best with and without keywork respectively. Taking the average of both (with and without keyword) t-SNE performs best.

Overall, in our experiments we observe RoBERTa, t-SNE, and K-Medoids to be the effective combination for trend detection.

5.2.2 ***Experiment Objective 2:*** *What is an effective preprocessing approach for SQL queries?*

In the NLP, the positive affects of preprocessing of text on the final outcome has been widely explored [9, 54]. Our goal is to explore and analyze preprocessing on SQL queries. We design an experiment for this analysis. We pre-process workload by removing SQL keywords. We analyze pre-processing on `TPC-H Random`, `TPC-H Template`, and `IMDB` workloads. We compare this analysis with regular SQL queries (with keywords), as shown in Fig 7, 6. The visual representation for `TPC-H Template` queries has no major distinction. The both representation with and without keywords were similar in nature. In general all queries are equally set apart. Such a representation shows the queries are very different from each other. This analysis is validated from the actual queries. The workload designed by QGEN using `TPC-H Template` represents 22 different queries. They retrieve very different set of data and are quite different from each other. However, in the representation for `TPC-H Random`, we observe a few similarity. The data can be observed in clusters. When comparing cluster of queries, they are more evident after preprocessing. We can observe similar queries were group together after preprocessing, such as query [33,34] and [1,3]: However, that was not the case in the representation with keywords.

```
ID 33: SELECT COUNT(*) FROM lineitem WHERE L_ORDERKEY = 2533062
ID 34: SELECT COUNT(*) FROM lineitem WHERE L_ORDERKEY < 4839745

ID 1: SELECT COUNT(*) FROM lineitem WHERE L_SUPPKEY < 15859
ID 3: SELECT COUNT(*) FROM lineitem WHERE L_SUPPKEY < 16455
```

We also observe the same in `IMDB Dataset`. At a first glance, representation without keywords is spread out and similar queries are clustered, as contrary to representation with keywords. There are a few clusters that are consistent in both representations as well are shown in red boxes in Fig 7, 6. An SQL is a structured language and each query has keywords. The contextual meaning of SQL query can be defined using predicates and expression. The preprocessing step preserves and enhances the contextual meaning of the query by removing the redundant information from queries. On the other hand, when the keywords are considered part of the text thereby loosing the overall contextual meaning of the query. Example, consider SQL query [Q1: *SELECT COUNT(*) FROM lineitem WHERE L_ORDERKEY = 2533062*], this query is *looking for a the data related to a specific (= 2533062) order placed (L_ORDERKEY).* When we convert that SQL query to natural language we can observe the contextual meaning is in the predicates, and the SQL keywords play a lesser role. By removing keywords the query becomes [Q2: *lineitem L_ORDERKEY = 2533062*]. This preprocessing helps to reduce the redundant information and thereby improving the quality

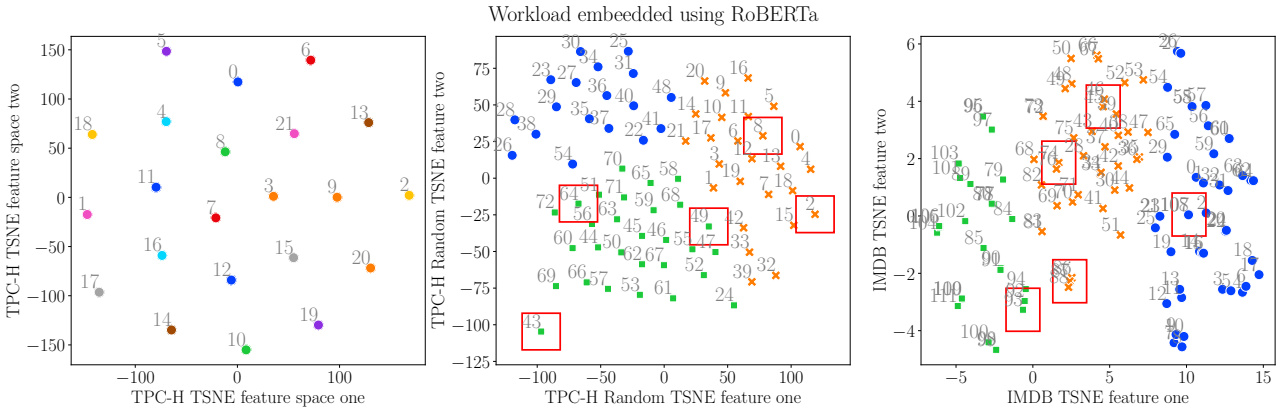

Figure 6: TPC-H Template, TPC-H Random, and IMDB workload representation. The numeric value represent serial number.

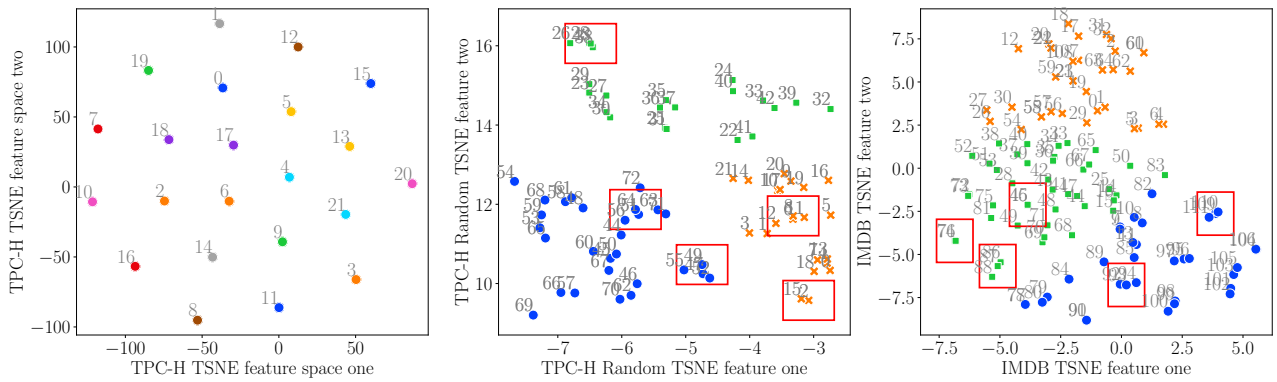

Figure 7: TPC-H Template, TPC-H Random, and IMDB workload representation with no keywords. The numeric value represent serial number.

of representation. With this observation we conclude that representation of SQL queries after preprocessing (removing keywords) is very effective.

5.2.3 ***Experiment Objective 3:*** *Is our proposed DQN algorithm with priority experience sweeping capable to learning the change in the environment?*

In this experiment, we aim to measure the online efficacy of the DQN algorithm for priority experience sweeping. To perform such evaluation we design an experiment with `IMDB Dataset` and `TPC-H Random` dataset. In the modern software systems, applications are regularly modified or added. Such modification changes the workload. We simulate and evaluate our framework in such a scenarios as well. We introduce future Groups of queries by merging and removing existing workloads. We categorize both datasets in three groups as described earlier in section 5.1. Our generated random queries from TPC-H consists of three groups, where Group 1 consists of queries only on single column, Group 2 with exactly two columns and Group 3 with exactly three columns. A sample queries from each group is shown below:

```
SELECT COUNT(*) FROM LINEITEM WHERE L_PARTKEY = 30217
SELECT COUNT(*) FROM LINEITEM WHERE L_ORDERKEY < 4741856 AND
    L_TAX = 0.02
SELECT COUNT(*) FROM LINEITEM WHERE L_SUPPKEY > 16616 AND L_TAX
    < 0.06 AND L_PARTKEY > 82374
```

The cumulative rewards graph for both datasets are shown in Fig 8 9 10 **??**, where each experiment has 15,000 episodes and Group 2 and 3 are introduced at 5,000 and 10,000 respectively.

In fig 8, we observe sharp decline in cumulative rewards at episode 5,000 and 10,000 and progressive nature later. This explains that the indexes chosen initially for Group 1 were not as effective for Group 2. This result aligns with the dataset as all three groups of IMDB dataset are entirely different from each other with no inter-dependency. However, in the TPC-H dataset groups have dependency on each other. This can be observed in Fig 9 10, where both Group 2 and 3 received a gain in performance from previous indexes. This experiment evaluates efficacy of our framework on two datasets in a few different scenarios.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose **Indexer++** a real-time solution for online index selection using pretrained NLP models and Deep Reinforcement Learning. We describe our approach for a change in workload pattern detection. We observe SQL representation to be effective using t-SNE, K-Medoids and RoBERTa pretrained model. We also

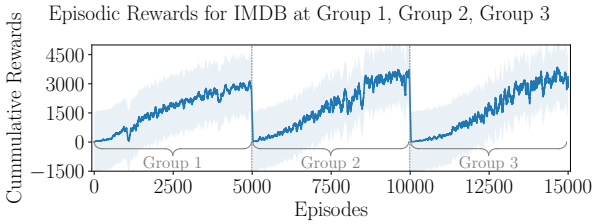

Figure 8: Cumulative Rewards by DQN agent on IMDB dataset. The dataset is introduced in three stages starting with Group 1 as initial workload then Group 2 and 3.

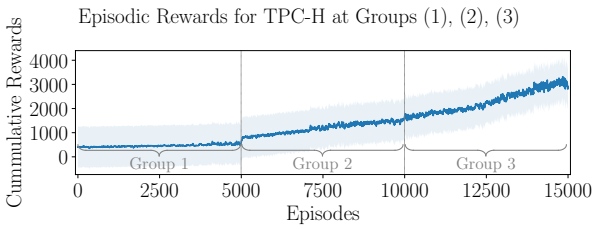

Figure 9: Cumulative Rewards by DQN agent on TPC-H dataset. The dataset is introduced in three stages starting with Group 1 as initial workload then Group 2 and 3.

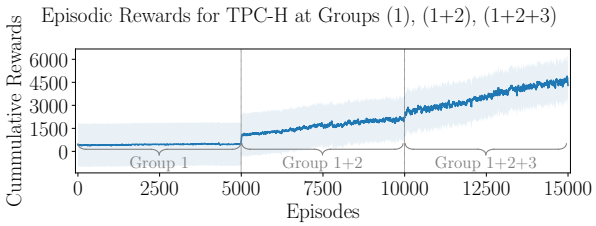

Figure 10: Cumulative Rewards by DQN agent on TPC-H dataset. The dataset is introduced in three stages starting with Group 1 as initial workload then combined with Group 2 and 3.

propose priority experience sweeping as an extension to DQN algorithm for generalization in an online environment. We evaluate our approach on three different datasets. In our experiments, we observe our proposed framework is able to solve all the existing challenges for an online index tuning namely, (1) Noise Resilience, (2) Overhead Cost, (3) Trend Detection Time, and (4) Response Time and respond to changing workload patterns by selecting optimal set of indexes.

In the future, we plan to extent our workload pattern detection to build a classifier to predict runtime, cardinality estimate and error. We also plan to analyze the affect of SQL operator in workload representation.

REFERENCES

[1] Joy Arulraj, Ran Xian, Lin Ma, and Andrew Pavlo. 2019. Predictive Indexing. *arXiv preprint arXiv:1901.07064* (2019).

[2] Bortik Bandyopadhyay, Pranav Maneriker, Vedang Patel, Saumya Yashmohini Sahai, Ping Zhang, and Srinivasan Parthasarathy. 2020. DrugDBEmbed: Semantic Queries on Relational Database using Supervised Column Encodings. *arXiv preprint arXiv:2007.02384* (2020).

[3] Debabrota Basu, Qian Lin, Weidong Chen, Hoang Tam Vo, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. 2015. Cost-Model Oblivious Database Tuning with Reinforcement Learning. In *Proceedings, Part I, of the 26th International Conference on Database and Expert Systems Applications - Volume 9261* (Valencia, Spain) *(DEXA 2015)*. Springer-Verlag, Berlin, Heidelberg, 253âĂŞ268. https://doi.org/10.1007/978-3-319-22849-5_18

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 8 (Aug. 2013), 1798–1828. https://doi.org/10.1109/TPAMI.2013.50

[5] Rajesh Bordawekar, Bortik Bandyopadhyay, and Oded Shmueli. 2017. Cognitive database: A step towards endowing relational databases with artificial intelligence capabilities. *arXiv preprint arXiv:1712.07199* (2017).

[6] Rajesh Bordawekar and Oded Shmueli. 2017. Using word embedding to enable semantic queries in relational databases. In *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*. 1–4.

[7] N. Bruno and S. Chaudhuri. 2007. An Online Approach to Physical Design Tuning. In *2007 IEEE 23rd International Conference on Data Engineering*. 826–835. https://doi.org/10.1109/ICDE.2007.367928

[8] N. Bruno and S. Chaudhuri. 2007. An Online Approach to Physical Design Tuning. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE Computer Society, Los Alamitos, CA, USA, 826–835. https://doi.org/10.1109/ICDE.2007.367928

[9] Jose Camacho-Collados and Mohammad Taher Pilehvar. 2018. On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 40–46.

[10] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1335âĂŞ1349. https://doi.org/10.1145/3318464.3389742

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

[12] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2978–2988.

[13] Sudipto Das, Miroslav Grbic, Igor Ilic, Isidora Jovandic, Andrija Jovanovic, Vivek R Narasayya, Miodrag Radulovic, Maja Stikic, Gaoxiang Xu, and Surajit Chaudhuri. 2019. Automatically indexing millions of databases in microsoft azure sql database. In *Proceedings of the 2019 International Conference on Management of Data*. 666–679.

[14] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[15] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2019. AI Meets AI: Leveraging Query Executions to Improve Index Recommendations *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1241âĂŞ1258. https://doi.org/10.1145/3299869.3324957

[16] Martin R. Frank, Edward Omiecinski, and Shamkant B. Navathe. 1992. Adaptive and Automated Index Selection in RDBMS. In *Proceedings of the 3rd International Conference on Extending Database Technology: Advances in Database Technology (EDBT '92)*. Springer-Verlag, Berlin, Heidelberg, 277âĂŞ292.

[17] Michael Günther. 2018. Freddy: Fast word embeddings in database systems. In *Proceedings of the 2018 International Conference on Management of Data*. 1817–1819.

[18] Michael Hammer and Arvola Chan. 1976. Index selection in a self-adaptive data base management system. In *Proceedings of the 1976 ACM SIGMOD international conference on Management of data*. 1–8.

[19] Michael Hammer and Arvola Chan. 1976. Index Selection in a Self-Adaptive Data Base Management System. In *Proceedings of the 1976 ACM SIGMOD International Conference on Management of Data* (Washington, D.C.) *(SIGMOD '76)*. Association for Computing Machinery, New York, NY, USA, 1âĂŞ8. https://doi.org/10.1145/509383.509385

[20] Matteo Hessel, Joseph Modayil, H. V. Hasselt, T. Schaul, Georg Ostrovski, W. Dabney, Dan Horgan, B. Piot, Mohammad Gheshlaghi Azar, and D. Silver. 2018. Rainbow: Combining Improvements in Deep Reinforcement Learning. *ArXiv* abs/1710.02298 (2018).

[21] Geoffrey E Hinton et al. 1986. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, Vol. 1. Amherst, MA, 12.

[22] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. 1999. Data clustering: a review. *ACM computing surveys (CSUR)* 31, 3 (1999), 264–323.

[23] Shrainik Jain, Bill Howe, Jiaqi Yan, and Thierry Cruanes. 2018. Query2Vec: An Evaluation of NLP Techniques for Generalized Workload Analytics. *arXiv preprint arXiv:1801.05613* (2018).

[24] Herald Kllapi, Ilia Pietri, Verena Kantere, and Yannis E Ioannidis. 2020. Automated Management of Indexes for Dataflow Processing Engines in IaaS Clouds.. In *EDBT*. 169–180.

[25] Piotr Kołaczkowski and Henryk Rybiński. 2009. *Automatic Index Selection in RDBMS by Exploring Query Execution Plan Space*. Springer Berlin Heidelberg, Berlin, Heidelberg, 3–24. https://doi.org/10.1007/978-3-642-02190-9_1

[26] J. Kossmann and R. Schlosser. 2019. A Framework for Self-Managing Database Systems. In *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*. 100–106.

[27] Tim Kraska, Mohammad Alizadeh, Alex Beutel, H. Ed Chi, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. 2019. SageDB - A Learned Database System. *CIDR* (2019).

[28] Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291* (2019).

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

[30] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[31] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. PMLR, 1188–1196.

[32] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (Nov. 2015), 204–215. https://doi.org/10.14778/2850583.2850594

[33] Omer Levy and Yoav Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the eighteenth conference on computational natural language learning*. 171–180.

[34] Omer Levy and Yoav Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the eighteenth conference on computational natural language learning*. 171–180.

[35] J. Lilleberg, Y. Zhu, and Y. Zhang. 2015. Support vector machines and Word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*. 136–140. https://doi.org/10.1109/ICCI-CC.2015.7259377

[36] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* (2017).

[37] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. 2873–2879.

[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[39] M. Luhring, K. Sattler, K. Schmidt, and E. Schallehn. 2007. Autonomous Management of Soft Indexes. In *2007 IEEE 23rd International Conference on Data Engineering Workshop*. 450–458. https://doi.org/10.1109/ICDEW.2007.4401028

[40] Mingbo Ma, Liang Huang, Bowen Zhou, and Bing Xiang. 2015. Dependency-based Convolutional Neural Networks for Sentence Embedding. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 174–179.

[41] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. 2019. Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning. arXiv:1911.04936 [cs.LG]

[42] Tomas Mikolov, Kai Chen, G. S. Corrado, and J. Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR*.

[43] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. 2018. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*. 9839–9849.

[44] Priscilla Neuhaus, Julia Couto, Jonatas Wehrmann, Duncan Dubugras Alcoba Ruiz, and Felipe Rech Meneguzzi. 2019. GADIS: A genetic algorithm for database index selection. In *The 31st International Conference on Software Engineering & Knowledge Engineering, 2019, Portugal.*

[45] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. 2017. Self-Driving Database Management Systems.. In *CIDR*, Vol. 4. 1.

[46] Wendel Góes Pedrozo, Júlio Cesar Nievola, and Deborah Carvalho Ribeiro. 2018. An Adaptive Approach for Index Tuning with Learning Classifier Systems on Hybrid Storage Environments. In *Hybrid Artificial Intelligent Systems*, Francisco Javier de Cos Juez, José Ramón Villar, Enrique A. de la Cal, Álvaro Herrero, Héctor Quintián, José António Sáez, and Emilio Corchado (Eds.). Springer International Publishing, Cham, 716–729.

[47] Gregory Piatetsky-Shapiro. 1983. The Optimal Selection of Secondary Indices is NP-Complete. *SIGMOD Rec.* 13, 2 (Jan. 1983), 72âĂŞ75.

[48] Z. Sadri, L. Gruenwald, and E. Leal. 2020. Online Index Selection Using Deep Reinforcement Learning for a Cluster Database. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*. 158–161. https://doi.org/10.1109/ICDEW49219.2020.00035

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

[50] Karl Schnaitter, Serge Abiteboul, Tova Milo, and Neoklis Polyzotis. 2006. Colt: continuous on-line tuning. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. 793–795.

[51] Karl Schnaitter, Neoklis Polyzotis, and Lise Getoor. 2009. Index Interactions in Physical Design Tuning: Modeling, Analysis, and Applications. *Proc. VLDB Endow.* 2, 1 (Aug. 2009), 1234âĂŞ1245. https://doi.org/10.14778/1687627.1687766

[52] Aliaksei Severyn and Alessandro Moschitti. 2015. Twitter Sentiment Analysis with Deep Convolutional Neural Networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Santiago, Chile) *(SIGIR '15)*. Association for Computing Machinery, New York, NY, USA, 959âĂŞ962. https://doi.org/10.1145/2766462.2767830

[53] Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. 2018. The Case for Automatic Database Administration using Deep Reinforcement Learning. *CoRR* abs/1801.05643 (2018). arXiv:1801.05643 http://arxiv.org/abs/1801.05643

[54] Xiaobing Sun, Xiangyue Liu, Jiajun Hu, and Junwu Zhu. 2014. Empirical Studies on the NLP Techniques for Source Code Data Preprocessing. In *Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies* (Nanjing, China) *(EAST 2014)*. Association for Computing Machinery, New York, NY, USA, 32âĂŞ39. https://doi.org/10.1145/2627508.2627514

[55] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[56] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[57] Gary Valentin, Michael Zuliani, Daniel C Zilio, Guy Lohman, and Alan Skelley. 2000. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of 16th International Conference on Data Engineering (Cat. no. 00CB37073)*. IEEE, 101–110.

[58] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-Scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1009âĂŞ1024. https://doi.org/10.1145/3035918.3064029

[59] Zhengtong Yan, Jiaheng Lu, Naresh Chainani, and Chunbin Lin. [n.d.]. Workload-Aware Performance Tuning for Autonomous DBMSs. *Machine learning* 13 ([n. d.]), 15.

[60] Shengyu Zhu, Ignavier Ng, and Zhitang Chen. 2020. Causal Discovery with Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.

CHAPTER 5

Conclusion and Future Work

A *Self Managing Database* eliminates the burden of managing and performing self-tuning of its parameters. They provide the most efficient way to store and retrieve data. In a step towards such databases we proposed and evaluate our frameworks *MANTIS* and *Indexer++* that are able to tune configurations by itself and also provide assistance to users by recommending indexes in real-time. Analyzing large data and millions of queries can be a challenging task. A *Database Assistant* can help us identify 'interesting' queries in a large corpus of queries, recommend sets of queries, and summarize a workload. We propose and evaluate *Query Enhancer* a contextual SQL query enhancer able to recommend interesting queries. To understand and link the relationship of tables in a database we propose *LinkSocial*. We perform extensive evaluations of our frameworks using multiple datasets and compare our results with existing state-of-the-art approaches. Our proposed frameworks out performs existing approaches.

In future, we plan to extent our work in a few ways. An exciting area of future research is to extent our workload pattern detection to predict run time, cardinality estimate and error from workload. Another area of exploration is to optimize several database parameters using reinforcement learning and eventually leading towards an autonomous self managing database. Our proposed *Query Enhancer* can be a extended as a platform for learning SQL, and this framework will be independent of platforms and can perform equally well across-platform. Entity Resolution on social media can be extended to study a user's behavior across platforms. To our knowledge such an analysis has not yet been studied.

REFERENCES

[1] J. Tan, T. Zhang, F. Li, J. Chen, Q. Zheng, P. Zhang, H. Qiao, Y. Shi, W. Cao, and R. Zhang, "Ibtune: Individualized buffer tuning for large-scale cloud databases," *Proc. VLDB Endow.*, vol. 12, no. 10, p. 1221–1234, Jun. 2019. [Online]. Available: https://doi.org/10.14778/3339490.3339503

[2] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 489–504. [Online]. Available: https://doi.org/10.1145/3183713.3196909

[3] B. Hilprecht, C. Binnig, and U. Röhm, "Towards learning a partitioning advisor with deep reinforcement learning," in *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, ser. aiDM '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3329859.3329876

[4] R. Marcus and O. Papaemmanouil, "Deep reinforcement learning for join order enumeration," in *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, ser. aiDM'18. New York, NY, USA: ACM, 2018, pp. 3:1–3:4. [Online]. Available: http://doi.acm.org/10.1145/3211954.3211957

[5] X. Zhou, C. Chai, G. Li, and J. SUN, "Database meets artificial intelligence: A survey," *IEEE Transactions on Knowledge Data Engineering*, no. 01, pp. 1–1, may 5555.

[6] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala, "Database tuning advisor for microsoft sql server 2005," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 930–932.

[7] M. Hammer, "Self-adaptive automatic data base design," in *Proceedings of the June 13-16, 1977, national computer conference*, 1977, pp. 123–129.

[8] M. Hammer and A. Chan, "Index selection in a self-adaptive data base management system," in *Proceedings of the 1976 ACM SIGMOD international conference on Management of data*, 1976, pp. 1–8.

[9] M. Hammer and B. Niamir, "A heuristic approach to attribute partitioning," in *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, 1979, pp. 93–101.

[10] M. Frank, E. Omiecinski, and S. Navathe, "Adaptive and automative index selection in rdbms," in *Proceedings of EDBT*, vol. 92.

[11] G. Weikum, C. Hasse, A. Mönkeberg, and P. Zabback, "The comfort automatic tuning project," *Information systems*, vol. 19, no. 5, pp. 381–432, 1994.

[12] S. Chaudhuri and V. R. Narasayya, "An efficient, cost-driven index selection tool for microsoft sql server," in *VLDB*, vol. 97. Citeseer, 1997, pp. 146–155.

[13] S. Chaudhuri and V. Narasayya, "Physical database design assistant and wizard for sql server," Microsoft Research Technical report, in preparation, Tech. Rep., 1997.

[14] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback, "Self-tuning database technology and information services: from wishful thinking to viable engineering," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 20–31.

[15] W. Tian, P. Martin, and W. Powley, "Techniques for automatically sizing multiple buffer pools in db2," in *Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*, 2003, pp. 294–302.

[16] G. Valentin, M. Zuliani, D. C. Zilio, G. Lohman, and A. Skelley, "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.

[17] D. C. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden, "Db2 design advisor: integrated automatic physical database design," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004, pp. 1087–1097.

[18] A. J. Storm, C. Garcia-Arellano, S. S. Lightstone, Y. Diao, and M. Surendra, "Adaptive self-tuning memory in db2," in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 1081–1092.

[19] B. Dageville and M. Zait, "Sql memory management in oracle9i," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 962–973.

[20] B. Dageville and K. Dias, "Oracle's self-tuning architecture and solutions."

[21] A. Pavlo, M. Butrovich, A. Joshi, L. Ma, P. Menon, D. V. Aken, L. J. Lee, and R. Salakhutdinov, "External vs. internal: An essay on machine learning agents for autonomous database management systems," *IEEE Data Eng. Bull.*, vol. 42, pp. 32–46, 2019.

[22] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1009–1024. [Online]. Available: https://doi.org/10.1145/3035918.3064029

[23] B. Zhang, D. Van Aken, J. Wang, T. Dai, S. Jiang, J. Lao, S. Sheng, A. Pavlo, and G. J. Gordon, "A demonstration of the ottertune automatic database management

system tuning service," *Proc. VLDB Endow.*, vol. 11, no. 12, p. 1910–1913, Aug. 2018. [Online]. Available: https://doi.org/10.14778/3229863.3236222

[24] T. Kraska, M. Alizadeh, A. Beutel, H. E. Chi, A. Kristo, G. Leclerc, S. Madden, H. Mao, and V. Nathan, "Sagedb - a learned database system," *CIDR*, 2019.

[25] J. Kossmann and R. Schlosser, "A framework for self-managing database systems," in *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*, 2019, pp. 100–106.

[26] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah *et al.*, "Self-driving database management systems," vol. 4, 2017, p. 1.

[27] B. Ding, S. Das, R. Marcus, W. Wu, S. Chaudhuri, and V. R. Narasayya, "Ai meets ai: Leveraging query executions to improve index recommendations," ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1241–1258. [Online]. Available: https://doi.org/10.1145/3299869.3324957

[28] P. Neuhaus, J. Couto, J. Wehrmann, D. D. A. Ruiz, and F. R. Meneguzzi, "Gadis: A genetic algorithm for database index selection," in *The 31st International Conference on Software Engineering & Knowledge Engineering, 2019, Portugal.*, 2019.

[29] X. Yu, G. Li, C. Chai, and N. Tang, "Reinforcement learning with tree-lstm for join order selection," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1297–1308.

[30] I. Trummer, S. Moseley, D. Maram, S. Jo, and J. Antonakakis, "Skinnerdb: Regret-bounded query evaluation via reinforcement learning," *Proc. VLDB Endow.*, vol. 11, no. 12, pp. 2074–2077, Aug. 2018. [Online]. Available: https://doi.org/10.14778/3229863.3236263

[31] R. Marcus and O. Papaemmanouil, "Towards a hands-free query optimizer through deep learning," *CoRR*, vol. abs/1809.10212, 2018. [Online]. Available: http://arxiv.org/abs/1809.10212

[32] S. Krishnan, Z. Yang, K. Goldberg, J. M. Hellerstein, and I. Stoica, "Learning to optimize join queries with deep reinforcement learning," *CoRR*, vol. abs/1808.03196, 2018. [Online]. Available: http://arxiv.org/abs/1808.03196

[33] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi, "Learning state representations for query optimization with deep reinforcement learning," in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, 2018, pp. 1–4.

[34] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica, "Learning to optimize join queries with deep reinforcement learning," 2018.

[35] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li, "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 415–432. [Online]. Available: https://doi.org/10.1145/3299869.3300085

[36] G. Li, X. Zhou, S. Li, and B. Gao, "Qtune: A query-aware database tuning system with deep reinforcement learning," *Proc. VLDB Endow.*, vol. 12, no. 12, p. 2118–2130, Aug. 2019. [Online]. Available: https://doi.org/10.14778/3352063.3352129

[37] G. C. Durand, M. Pinnecke, R. Piriyev, M. Mohsen, D. Broneske, G. Saake, M. S. Sekeran, F. Rodriguez, and L. Balami, "Gridformation: Towards self-driven online data partitioning using reinforcement learning," in *aiDM'18*, 2018.

[38] Z. Yang, B. Chandramouli, C. Wang, J. Gehrke, Y. Li, U. F. Minhas, P.-r. Larson, D. Kossmann, and R. Acharya, "Qd-tree: Learning data layouts for big data analytics," in *Proceedings of the 2020 ACM SIGMOD International*

Conference on Management of Data, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 193–208. [Online]. Available: https://doi.org/10.1145/3318464.3389770

[39] D. Basu, Q. Lin, W. Chen, H. T. Vo, Z. Yuan, P. Senellart, and S. Bressan, "Cost-model oblivious database tuning with reinforcement learning," in *Proceedings, Part I, of the 26th International Conference on Database and Expert Systems Applications - Volume 9261*, ser. DEXA 2015. Berlin, Heidelberg: Springer-Verlag, 2015, p. 253–268. [Online]. Available: https://doi.org/10.1007/978-3-319-22849-5_18

[40] A. Sharma, F. M. Schuhknecht, and J. Dittrich, "The case for automatic database administration using deep reinforcement learning," *CoRR*, vol. abs/1801.05643, 2018. [Online]. Available: http://arxiv.org/abs/1801.05643

[41] J. Welborn, M. Schaarschmidt, and E. Yoneki, "Learning index selection with structured action spaces," *arXiv preprint arXiv:1909.07440*, 2019.

[42] G. P. Licks, J. C. Couto, P. de Fátima Miehe, R. De Paris, D. D. Ruiz, and F. Meneguzzi, "Smartix: A database indexing agent based on reinforcement learning," *Applied Intelligence*, pp. 1–14, 2020.

[43] H. Kllapi, I. Pietri, V. Kantere, and Y. E. Ioannidis, "Automated management of indexes for dataflow processing engines in iaas clouds." in *EDBT*, 2020, pp. 169–180.

[44] M. Luhring, K. Sattler, K. Schmidt, and E. Schallehn, "Autonomous management of soft indexes," in *2007 IEEE 23rd International Conference on Data Engineering Workshop*, 2007, pp. 450–458.

[45] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis, "Colt: Continuous on-line tuning," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 793–795. [Online]. Available: https://doi.org/10.1145/1142473.1142592

[46] W. G. Pedrozo, J. C. Nievola, and D. C. Ribeiro, "An adaptive approach for index tuning with learning classifier systems on hybrid storage environments," in *Hybrid Artificial Intelligent Systems*, F. J. de Cos Juez, J. R. Villar, E. A. de la Cal, Á. Herrero, H. Quintián, J. A. Sáez, and E. Corchado, Eds. Cham: Springer International Publishing, 2018, pp. 716–729.

[47] S. Das, M. Grbic, I. Ilic, I. Jovandic, A. Jovanovic, V. R. Narasayya, M. Radulovic, M. Stikic, G. Xu, and S. Chaudhuri, "Automatically indexing millions of databases in microsoft azure sql database," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 666–679. [Online]. Available: https://doi.org/10.1145/3299869.3314035

[48] Z. Sadri, L. Gruenwald, and E. Leal, "Online index selection using deep reinforcement learning for a cluster database," in *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, 2020, pp. 158–161.

[49] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li, "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 415–432. [Online]. Available: https://doi.org/10.1145/3299869.3300085

[50] M. Olma, M. Karpathiotakis, I. Alagiannis, M. Athanassoulis, and A. Ailamaki, "Slalom: Coasting through raw data via adaptive partitioning and indexing," *Proc. VLDB Endow.*, vol. 10, no. 10, p. 1106–1117, Jun. 2017. [Online]. Available: https://doi.org/10.14778/3115404.3115415

[51] S. Chaudhuri, A. K. Gupta, and V. Narasayya, "Compressing sql workloads," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 2002, pp. 488–499.

[52] P. Kołaczkowski, "Compressing very large database workloads for continuous online index selection," in *International Conference on Database and Expert Systems Applications*. Springer, 2008, pp. 791–799.

[53] B. Babcock, S. Chaudhuri, and G. Das, "Dynamic sample selection for approximate query processing," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 539–550. [Online]. Available: https://doi.org/10.1145/872757.872822

[54] S. Jain, B. Howe, J. Yan, and T. Cruanes, "Query2vec: An evaluation of nlp techniques for generalized workload analytics," *arXiv preprint arXiv:1801.05613*, 2018.

[55] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[56] P. Wang, T. Shi, and C. K. Reddy, "Text-to-sql generation for question answering on electronic medical records," in *Proceedings of The Web Conference 2020*, 2020, pp. 350–361.

[57] S. E. Whang, D. Marmaros, and H. Garcia-Molina, "Pay-as-you-go entity resolution," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 1111–1124, 2012.

[58] J. Cai and M. Strube, "End-to-end coreference resolution via hypergraph partitioning," in *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, 2010, pp. 143–151.

[59] Y. Qian, Y. Hu, J. Cui, Q. Zheng, and Z. Nie, "Combining machine learning and human judgment in author disambiguation," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 1241–1246.

[60] K. Shu, S. Wang, J. Tang, R. Zafarani, and H. Liu, "User identity linkage across online social networks: A review," *Acm Sigkdd Explorations Newsletter*, vol. 18, no. 2, pp. 5–17, 2017.

[61] S. Liu, S. Wang, F. Zhu, J. Zhang, and R. Krishnan, "Hydra: Large-scale social identity linkage via heterogeneous behavior modeling," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 51–62.

[62] H. Zhang, M.-Y. Kan, Y. Liu, and S. Ma, "Online social network profile linkage," in *Asia Information Retrieval Symposium*. Springer, 2014, pp. 197–208.

[63] W. W. Cohen and J. Richman, "Learning to match and cluster large high-dimensional data sets for data integration," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 475–480.

[64] P. Jain, P. Kumaraguru, and A. Joshi, "@ i seek'fb. me' identifying users across multiple online social networks," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 1259–1268.

[65] Y. Li, Z. Zhang, Y. Peng, H. Yin, and Q. Xu, "Matching user accounts based on user generated content across social networks," *Future Generation Computer Systems*, vol. 83, pp. 104–115, 2018.

[66] A. Nunes, P. Calado, and B. Martins, "Resolving user identities over social networks through supervised learning and rich similarity features," in *Proceedings of the 27th Annual ACM symposium on applied computing*, 2012, pp. 728–729.

[67] J. Liu, F. Zhang, X. Song, Y.-I. Song, C.-Y. Lin, and H.-W. Hon, "What's in a name? an unsupervised approach to link users across communities," in *Proceedings of the sixth ACM international conference on Web search and data mining*, 2013, pp. 495–504.

[68] R. Zafarani and H. Liu, "Connecting corresponding identities across communities." *ICWSM*, vol. 9, pp. 354–357, 2009.

[69] Q. Ma, H. H. Song, S. Muthukrishnan, and A. Nucci, "Joining user profiles across online social networks: From the perspective of an adversary," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2016, pp. 178–185.

[70] R. Zafarani and H. Liu, "Connecting users across social media sites: a behavioral-modeling approach," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 41–49.

[71] X. Mu, F. Zhu, E.-P. Lim, J. Xiao, J. Wang, and Z.-H. Zhou, "User identity linkage by latent user space modelling," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1775–1784.

[72] G. Silvestri, J. Yang, A. Bozzon, and A. Tagarelli, "Linking accounts across social networks: the case of stackoverflow, github and twitter." in *KDWeb*, 2015, pp. 41–52.

[73] G. Quercini, N. Bennacer, M. Ghufran, and C. N. Jipmo, "Liaison: reconciliation of individuals profiles across social networks," in *Advances in Knowledge Discovery and Management*. Springer, 2017, pp. 229–253.