# BRIDGING DATA MANAGEMENT AND MACHINE LEARNING: CASE STUDIES ON INDEX, QUERY OPTIMIZATION, AND DATA ACQUISITION

YIFAN LI

A DISSERTATION SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO
JUNE 2022

# Abstract

Data management tasks and techniques can be observed in a variety of real world scenarios, including web search, business analysis, traffic scheduling, and advertising, to name a few. While data management as a research area has been studied for decades, recent breakthroughs in Machine Learning (ML) provide new perspectives to define and tackle problems in the area, and at the same time, the wisdom integrated in data management techniques also greatly helps to accelerate the advancement of Machine Learning. In this work, we focus on the intersection area of data management and Machine Learning, and study several important, interesting, and challenging problems. More specifically, our work mainly concentrates on the following three topics: (1) leveraging the ability of ML models in capturing data distribution to design lightweight and data-adaptive indexes and search algorithms to accelerate similarity search over large-scale data; (2) designing robust and trustworthy approaches to improve the reliability of both conventional query optimizer and learned query optimizer, and boost the performance of DBMS; (3) developing data management techniques with statistical guarantees to acquire the most useful training data for ML models with a budget limitation, striving to maximize the accuracy of the model. We conduct detailed theoretical and empirical study for each topic, establishing these fundamental problems as well as developing efficient and effective approaches for the tasks.

# Acknowledgements

Working towards a Ph.D. degree is a long, challenging, but very rewarding process. Standing here, I feel deeply grateful for all the help, encouragement, and kindness I have received along the way.

First of all, I would like to express my sincere gratitude to my supervisor, Prof. Xiaohui Yu, for his guidance. Prof. Yu has been my Master degree supervisor as well, and in the past six years, Prof. Yu has mentored me with great passion and patience to help me become a successful researcher and encourage me to pursue excellence in my life.

I would also like to thank Prof. Nick Koudas for the valuable help and advice he has generously provided in the past four years. We cooperated together on all the works I conducted during my Ph.D. study, and Prof. Koudas has influence me greatly with his enthusiasm to work and wisdom towards life.

I offer my gratitude to my supervisory committee members, Prof. Aijun An and Prof. Manos Papagelis, and my examination committee members, Prof. Arik Senderovich, Prof. Fei Chiang, and Prof. Ling Jiang, for providing me with insightful suggestions to improve the dissertation.

I would like to thank my parents for their endless love and support. The work cannot be done without you, and I would like to dedicate this work with you.

Last but not least, I would like to thank my fiancee, Jingxiao. Over the past several years, she has been an excellent partner and friend, and the most motivated and passionate teammate whom I feel both lucky and honoured to grow together with.

# Contents

# List of Tables

# List of Figures

# 1 Introduction

## 1.1 Background and Motivation

Data management [149, 67] generally refers to the problems of processing, organizing, and retrieving data to improve the efficiency of various types of tasks such as query answering and similarity search, and is the heart of many modern applications such as databases and search engine. Although researchers have studied data management-related problems for decades, the unprecedented amount and speed of data being collected, processed, and utilized nowadays bring new challenges to data management. The era of big data calls for more efficient, scalable, and robust data management techniques in a wide variety of real-world scenarios.

Machine Learning (ML) [75], especially Deep Learning (DL) [52], draws ever-rising academic and industrial attention in the past decade. Recent advances in Machine Learning theories and frameworks have made it possible to use ML techniques to assist humans in varying tasks such as driving and surgery. Development in Machine Learning-related hardware such as GPU and TPU, and platforms such as Microsoft Azure and Amazon AWS have also enabled the wide deployment of Machine Learning techniques.

Although the main breakthroughs of Machine Learning first take place in areas such as Computer Vision (CV) [47] and Natural Language Processing (NLP) [102], the power of Machine Learning has facilitated the progress of data management in the past several years as well. Researchers have explored the possibility of using Machine Learning techniques to solve conventional and fundamental data management problems such as indexing, and observed promising and exciting results. Machine Learning enables researchers to model and tackle data management problems from different angles and creates a potentially huge

opportunity for a revolution in the data management area.

On the other hand, data management methods also benefit the advancement of Machine Learning [18, 161]. The ability of data management techniques in data cleaning, data augmentation, and data acquisition makes it possible to apply ML models to scenarios with insufficient training data or where training efficiency is a concern, and has significantly accelerated the real-world deployment of Machine Learning techniques.

## 1.2 Our Research Objective

Despite the promising research opportunities and the efforts researchers have made in the intersection area between data management and Machine Learning, there are certain challenges in applying ML techniques to data management tasks and vice versa. First, ML techniques, especially deep learning-related techniques, are known to incur large training and inference costs, while efficiency is one of the main concerns of data management tasks [20]. Second, the behaviour of ML models is usually unpredictable and the errors are unbounded, while in practice many data management applications require certain guarantees regarding the performance and accuracy [171]. Third, data management tasks usually involve heterogeneous data with unknown correlations, and proper embedding of such data which preserves necessary information is difficult [152].

**In this work, we contribute to bridging data management and Machine Learning, propose solutions that can help to overcome the above-mentioned challenges, and deeply investigate how data management and Machine Learning tasks can benefit from each other.** While the study in related areas is still in the preliminary state and many important research problems remain unsolved, **we choose three very fundamental and interesting directions to explore in detail in this work, namely, index, query optimization, and data acquisition.** More specifically, index and query optimization are two (perhaps the most) important problems researchers have been working on in the area of data management to reduce the latency of data retrieval and processing, and data acquisition (e.g., acquiring useful data for training) is one of the bottleneck tasks in training Machine Learning models. As will be shown later in this thesis, ML techniques help

to reshape the perspective we view and solve index and query optimization tasks, providing a completely new category of effective approaches; and data acquisition techniques that are well-studied in the data management area benefit Machine Learning, greatly reducing the time and monetary cost of building ML models.

## 1.3 Overview of Related Research Topics

In this section we present a high-level overview of the important research topics in the intersection area of data management and Machine Learning, to provide the reader with a clear picture of the academic interests and industrial demands in the area, and a more detailed literature review can be located in Section 2. Below we mainly focus on the three topics related to the questions we investigate in this work: (1) ML-based indexing and searching, (2) ML-based query optimization, and (3) ML model-oriented data exchange.

### 1.3.1 ML-based Indexing and Searching

Data indexing and searching, one of the fundamental tasks in the data management area, aim to efficiently locate and retrieve the data specified by a query predicate. While existing algorithms need to sort and partition data to reduce search overhead, ML techniques provide a new perspective to view and solve the problem. More specifically, researchers have proposed to view index as a ML model, with the input being the query predicate and the output being the location of the requested data, and thus the data finding process can be viewed as model inference. It has been shown that ML models can effectively capture the mapping from a query predicate to data locations. Compared with conventional techniques, ML-based indexes are usually more lightweight and accurate.

### 1.3.2 ML-based Query Optimization

Query optimization is one of the most important and challenging tasks in Database Management Systems (DBMS). The objective of query optimization is to identify the most efficient one from a collection of candidate plans that can be executed to answer a particular query. Previous query optimizers have various hand-crafted rules for plan selection, which usually

fail to identify the optimal plan. The major sub-task of query optimization, the plan cost estimation, is essentially a regression task, and thus we can leverage ML model to directly learn the mapping from a query plan to its execution time, which would help to identify the fastest plan.

### 1.3.3 ML Model-oriented Data Exchange

Machine Learning models are usually data-hungry and require sufficient high-quality data for performance improvement. In practice, the model owner usually needs to acquire data from other parties, such as e-commerce companies with customer shopping records, at some (usually monetary) expense. The wisdom of data management in tasks such as data exchange would greatly help to design a healthy and flawless mechanism for data providing and acquisition. Relevant techniques in data management can also help to maximize the data owner's revenue in providing the data, or minimize the model owner's expense in acquiring the data to reach a desired level of model accuracy.

## 1.4 Problems Studied in This Work

As mentioned in previous sections, researches in the intersection area of data management and Machine Learning is at the early stage and many interesting questions remain to be solved. In this work we select three of the most fundamental and important tasks in the area and propose our solutions, including data indexing, query optimization, and data acquisition. The former two tasks are typical data management tasks, and we investigate how ML techniques can be utilized to better solve the problems, and data acquisition or acquiring useful training data is one of the bottleneck problems of ML, which we believe would greatly benefit from data management techniques. In this section we give a brief overview of the three problems studied in this work.

### 1.4.1 Learning-based Set Similarity Search

Set similarity search is a problem of central interest to a wide variety of applications such as data cleaning and web search. Past approaches to set similarity search utilize either heavy

indexing structures, incurring large search costs or indexes that produce large candidate sets. We design a learning-based exact set similarity search approach, LES$^3$. Our approach first partitions sets into groups, and then utilizes a lightweight bitmap-like indexing structure, called token-group matrix (TGM), to organize groups and prune out candidates given a query set. In order to optimize pruning using the TGM, we analytically investigate the optimal partitioning strategy under certain distributional assumptions. Using these results, we then design a learning-based partitioning approach called L2P and an associated data representation encoding, PTR, to identify the partitions. We conduct extensive experiments on real and synthetic datasets to fully study LES$^3$, establishing the effectiveness and superiority over other applicable approaches. More details regarding the index structures, algorithms, and experiment results are provided in Section 3.

### 1.4.2 Learning-assisted Query Plan Selection

Plan selection, as the core task of query optimization, is known to be difficult in practice especially when the tables are co-related and the queries involve multiple tables/attributes. Besides conventional cost estimators (such as cardinality estimator plus cost model) [180], recently researchers show great interests in using ML-based technique to assist plan selection and observe its superiority over conventional techniques [113, 116, 153]. However, as pointed out in [171], the estimation of both conventional cost estimators and ML-based estimators are not always trustworthy, and using a single estimated cost for plan selection sometimes leads to arbitrarily bad query performance. Therefore, in this work, we complement existing plan selection techniques by constructing the execution time distribution for various plans and leveraging the distribution for more controllable plan selection. To construct the execution time distribution, we design a novel approach based on a robust statistical tool named conformal prediction. Our design can be applied to different types of cost estimators including conventional DBMS cost estimators and learned cost estimators, and can be easily integrated into an existing DBMS. Based on the execution time distribution, we design several intuitive and fundamental objectives regarding the query performance such as time limits. Details of building the utilizing the execution time distribution can be found in Section 4.

### 1.4.3 Data Acquisition for Improving Machine Learning Models

The vast advances in Machine Learning (ML) over the last ten years have been powered by the availability of suitably prepared data for training purposes. The future of ML-enabled enterprises hinges on data. As such, there is already a vibrant market offering data annotation services to tailor sophisticated ML models.

Inspired by the recent vision of online data markets and associated market designs, we present research on the practical problem of obtaining data in order to improve the accuracy of ML models. We consider an environment in which consumers query for data to enhance the accuracy of their models and data providers who possess data make them available for training purposes. We first formalize this interaction process laying out the suitable framework and associated parameters for data exchange. We then propose two data acquisition strategies that consider a trade-off between *exploration* during which we obtain data to learn about the distribution of a provider's data and *exploitation* during which we optimize our data inquiries utilizing the gained knowledge. In the first strategy, *Estimation and Allocation* (EA), we utilize queries to estimate the utilities of various predicates while learning about the distribution of the provider's data; then we proceed to the allocation stage in which we utilize those learned utility estimates to inform our data acquisition decisions. The second algorithmic proposal, named *Sequential Predicate Selection* (SPS), utilizes a sampling strategy to explore the distribution of the provider's data, adaptively investing more resources to parts of the data space that are statistically more promising to improve overall model accuracy. Details regarding the problem and approaches are provided in Section 5.

## 1.5 Summary of Contributions

In this thesis we contribute to bridging data management with Machine Learning and investigate three fundamental problems in the intersection area. Our contributions can be summarized as follows:

1. We present a thorough review of recent works related to the intersection of data management and Machine Learning, together with a detailed analysis of the advantages

and limitations of each approach, and point out multiple promising research directions that worth further investigation.

2. We study the problem of designing learned indexing and searching algorithms to accelerate similarity search over large-scale set data. More specifically, we propose LES$^3$which follows a filter-and-verify methodology and first partition sets into disjoint groups using a learned approach L2P, and then index each group with a lightweight bitmap-like structure TGM. The main component of L2P is a ML model whose objective is to assign each set into a group so that the pruning power of the TGM can be maximized. We conduct extensive experiments on real and synthetic datasets to show the effectiveness of the proposed approach in improving search efficiency and reducing storage overhead.

3. We devise a novel and robust approach for plan selection that leverages the execution time distribution of query plans. We adapt a well-designed statistical tool named conformal prediction to construct the execution time distribution for new plans, with very limited assumptions regarding the data and workload distribution. The designed execution time distribution construction method works for both conventional query optimizers and modern learned query optimizers. We then develop several intuitive and fundamental objectives that the DBMS may possess when asking a query, such as the maximal query answering latency, and design plan selection strategies leveraging the execution time distribution to fulfill each of the objectives. We also present a detailed study about how to integrate the new design into DBMS with very minor extra overhead. Extensive experiments with different DBMSs, query optimizers, and benchmarks prove the effectiveness of our proposal.

4. We study the problem of obtaining the most useful data to improve the accuracy of ML models given a fixed budget (limiting the amount of data that can be acquired). We design a practical data market to allow the data consumer to acquire desired data from the data provider by specifying the properties of the required data using queries. We further propose two approaches for the data consumer to acquire the most useful data with a limited budget, which effectively balance between exploration during which

the data consumer acquires data to learn the data provider's data distribution and identify the queries that specify the most useful data, and exploitation, during which the data consumer leverages the gained knowledge to acquire the most useful data. The proposed methods are thoroughly testified with various datasets and ML models and demonstrate stable and superior performance across different settings.

5. Besides the problems analyzed and studied in detail in this thesis, we also list multiple research topics that we believe are both interesting and important to investigate as future works. These topics and problems would help to accelerate the development of this novel research area as well as benefit a variety of real applications and systems.

## 1.6 Published and Under-review Works

The dissertation is based on the following published or under-review works:

1. LES$^3$: Learning-based Exact Set Similarity Search

   **Yifan Li**, Xiaohui Yu, Nick Koudas. Proceedings of the VLDB Endowment, 2021

2. Data Acquisition for Improving Machine Learning Models

   **Yifan Li**, Xiaohui Yu, Nick Koudas. Proceedings of the VLDB Endowment, 2021

3. dbET: Execution Time Distribution-based Plan Selection

   **Yifan Li**, Xiaohui Yu, Nick Koudas, Shu Lin, Calvin Sun, Chong Chen. Under review

My other works during Ph.D. study are listed below:

1. Top-$k$ queries over Digital Traces

   **Yifan Li**, Xiaohui Yu, Nick Koudas. Proceedings of the 2019 International Conference on Management of Data (SIGMOD'19)

2. FILM: A Fully Learned Index for Larger-than-memory Databases

   Chaohong Ma, Xiaohui Yu, **Yifan Li**, Xiaofeng Meng, Aishan Maoliniyazi. Under Review

3. Distributed Processing of k Shortest Path Queries over Dynamic Road Networks

   Ziqiang Yu, Xiaohui Yu, Nick Koudas, Yang Liu, **Yifan Li**, Yueting Chen, Dingyu Yang. Proceedings of the 2020 International Conference on Management of Data (SIGMOD'20)

# 2 Literature Review

In this section, we present the literature in the intersection area of data management and Machine Learning in a broad sense, to provide an overview of the research area. Detailed related works regarding each research problem introduced in Section 1.4 can be located in the corresponding sections.

## 2.1 Similarity Search

Similarity search in data management is usually accelerated by indexes, which are data structures and algorithms that organize data in a way to speed-up look-up operations. Famous examples of indexes include B-tree (for one-dimensional data) and R-tree (for high-dimensional data). Indexes serve as one of the fundamental components of data management and are always the focus of academic research as well as industrial application.

Over the past few years, researchers started to construct and optimize indexes with the help of Machine Learning techniques. More specifically, researchers strive to replace the underlying data structures and algorithms with ML models so that given a search query, the model can directly output the query results. Breakthrough is observed for tree-based index structures, hash-based structures, partitioning methods, etc. The major advantages of using ML-based indexes to replace conventional indexes include:

1. By adapting to the data and uncovering patterns in a specific problem instance (e.g., a specific workload or data distribution) instead of solving a general problem for every problem instance as conventional indexes do, ML-based indexes usually provide faster query answering;

2. By using non-linear functions to model the relation between a query and the expected

result, as opposed to using data structures such as nodes in B-tree, ML-based indexes usually incur orders of magnitude less space overhead.

In the following, we introduce some of the interesting explorations in applying ML techniques for data indexing, showcase their advantages, and analyze their shortcomings[34, 45, 50, 90, 92, 99, 104, 114, 124, 135, 174, 189, 186].

### 2.1.1 Exact Search Approaches

The most important index for exact similarity search is tree structures, including B-tree, R-tree, and their variants. Tree indexes are designed for fast key look-up. We illustrate how tree indexes work with the example of a B-tree, which organizes data with a sorted array. Given a search key $K$, the search starts at the root and is repeatedly routed to a child node according to certain comparison operations until a leaf node is reached, where the data corresponding to $K$ is expected to be present.

Since the functionality of B-tree is to map a search key into a position in a sorted array, one can replace the tree with a regression model with the search key being the input and the position being the output.

**RMI.** The regression task of mapping a search key to a position can be tackled by learning the cumulative density function (CDF) of the data. Kraska et al. [90] propose a framework named Recursive Model Index (RMI), which consists of a collection of light-weight linear models, to learn the CDF and replace the tree structure. A search key is first fed into Model 1.1 and recursively routed to a child model rather than a child node, until a model at the bottom level is reached and a position is computed.

However, by learning the CDF with models, RMI may cause errors when making predictions for a given search key. For example, the predicted position may be 10 while the true position is 15. In order to guarantee correctness, RMI first makes predictions for all values in the array, let $D$ be the maximal distance between the predicted position and the true position of any value. Given an arbitrary search key $K$, suppose the predicted position is $P_K$, then values between position $P_K - D$ and position $P_K + D$ are checked to match $K$. The search strategy immediately reflects a shortcoming of RMI, which is, it can only handle

static data and OLAP (online analytical processing) scenarios.

**Flood.** Nathan et al. [124] extends the work of RMI from one-dimensional data to multi-dimensional data and propose Flood, an in-memory read-optimized index. While the basis is also capturing the CDF of the underlying data, Flood is able to adaptively adjust the layout of the data to project skewed data distribution into a more uniform space so as to accelerate search.

**Alex.** Ding et al. [34] extend previous learned indexes that mainly focus on look-up and search operations, and propose Alex, which supports write operations and provides fast search for dynamic data. With Alex, keys are mapped into a data structure named "gapped array" where spaces between keys are preserved so that newly inserted data can be mapped into the correct position in the array without causing much data re-position cost.

### 2.1.2 Approximate Search Approaches

Approximate similarity search usually reles on space transformation and partitioning, which discretize the entire space into sub-spaces and limit the search to certain sub-spaces. An example is locality sensitive hashing (LSH), which maps similar data items to the same bucket with higher probability, and for a given query item which is hashed into some bucket, its similar items (e.g., $k$ nearest neighbors) can be located in the same bucket. The two requirements for space partitioning methods are:

1. Fast to compute. Since the technique is mainly used to provide approximate answers for large datasets, search efficiency is one of the major concerns.

2. Accurate. Although the results are allowed to be approximate, high accuracy is always a desired property. High accuracy refers to (1) the true nearest neighbors have a high chance to be in the same sub-space as the query item, and (2) the similarity of approximate results is not far lower than the true results.

Researchers have explored to use ML-based techniques for space partitioning and observed that with these techniques both properties are likely to be better satisfied due to the fast inference of ML models and its ability in adapting to a certain data distribution.

**Space transformation.** Most indexing structures prefer uniformity of the data to provide faster search. Sablayrolles et al. [140] propose to map data points from the original space into a latent space where points are uniformly distributed while preserving the neighborhood relations between points to accelerate similarity search. In order to achieve the two properties (uniformity and neighbor-preserving) in the latent space, the authors design a loss function that integrates and naturally balances between an entropic regularizer to spread out points uniformly and a triplet rank-preserving loss which preserves the relative distance between points. Reshaping the data to uniform data would benefit a large family of approximate similarity search algorithms.

**Space partitioning.** Dong et al. [36] model the problem of space partitioning as a process of balanced graph partitioning followed by supervised classification. More specifically, the authors first construct a $k$NN graph for a given set of data points, i.e., view each point as a vertex and link each vertex with its $k$ nearest neighbors, and then adopt a graph partitioner to cut the $k$NN graph into sub-graphs, with the number of edges crossing different sub-graphs minimized. A classifier is then trained with points as the input and the corresponding sub-graph id as the labels. Given a new data point, the classifier maps it into a sub-graph, and its (approximate) nearest neighbors are identified from the points contained in this sub-graph.

### 2.1.3 Techniques for Set Similarity Search

In this section, we introduce the techniques designed for set similarity search, including conventional techniques and ML-based techniques, which are closely related to the problem studied in Section 3.

#### 2.1.3.1 Conventional Approaches

**Set Similarity Search Indexes**. The problem of processing set similarity queries, including set similarity search [83, 193, 194], i.e., find items in a collection that are similar to a query item, and set similarity joins [28, 30, 44, 111, 173], i.e., perform similarity search simultaneously for multiple query items, has attracted remarkable research interest recently. Zhang et al. [193, 194] propose to transform sets into scalars or vectors with the relative

distance between sets preserved, and organize the transformed sets with B+-trees or R-trees, which facilitate the use of tree-based branch-and-bound algorithms for similarity search. The major drawback of their work is that, as shown in the experiments, the tree structure can easily grow larger than the original data and thus using the index for filtering incurs a significant cost, especially when the index and the data are stored externally. Most prior research in the area of set similarity join focuses on threshold-join queries and follows the filter-and-verify framework. In the filter step, existing methods mainly adopt (1) prefix-based filters [14, 169, 182], based on the observation that if the similarity between two sets exceeds $\delta$, then they must share common token(s) in their prefixes of length $m$; and (2) partition-based filters [6, 29, 28, 181], which partition a set into several subsets so that two sets are similar only if they share a common subset.

#### 2.1.3.2 Set Embedding

Embedding sets and other entities consisting of discrete elements to facilitate adapting ML techniques for the task has been well-studied. The most natural way to represent such data types is $n$-hot encoding, but the resulting vectors are often very long and sparse. Dimensionality reduction techniques are used to compress the encoding vectors with different focuses: maximizing variances [131], preserving distances [13], solving the crowding problem [110], etc. Recent advances in document embedding, e.g., word2vec [119], BERT [32], also provide new perspectives to construct representations of sets. In our work, we propose a novel, lightweight, and effective set embedding approach PTR 3.5.3. Compared to these methods, PTR utilizes a very efficient method to produce relatively short representations and is optimized for the specific problem at hand.

### 2.1.4 Summary

Machine Learning for similarity search has received plenty of research interests over the past few years and researchers have explored the possibility and performance of using ML-based techniques to replace conventional hard-wired indexing structures and algorithms. Promising results are reported in a variety of settings. However, the area is still at the preliminary stage

and many questions remain open. For example, existing works are mainly index structure-oriented, i.e., use ML models to mimic the functionality of a specific index structure, but little work is task-oriented, i.e., design effective ML models for a given task such as string similarity search under edit distance. We believe task-oriented design would greatly facilitate the wide deployment of ML-based indexes in real world applications.

## 2.2 Query Optimization

Query optimization is the core to database management system (DBMS). A query optimizer is responsible for estimating the cost of each candidate query plan and selecting the optimal plan. In this section, we present works related to the major query optimization tasks, including cardinality estimation, join order selection, and trustworthy optimizers, with particular focus on the recent breakthroughs in ML-based query optimization.

### 2.2.1 Cardinality Estimation and Cost Estimation

The major challenge in estimating plan cost is to estimate the cardinality of sub-plans, i.e., the number of tuples in the intermediate results of a plan. Traditional methods for cardinality estimation utilize auxiliary structures such as histogram and generally assumes that different attributes in the database are independent. In recent years researchers model cardinality estimation as a regression problem which takes the input of an (encoded) query and output its estimated cardinality without making any assumption regarding the data distribution and achieve promising results [62, 64, 65, 72, 77, 85, 120, 152, 129, 163, 148, 167, 187, 188, 191].

ML models for cardinality estimation are mainly constructed in an off-line fashion, i.e., before query execution. Existing cardinality estimation models can be categorized into two groups: supervised cardinality estimation, which is query-oriented and built on a large number of training queries together with their cardinality; and unsupervised cardinality estimation, which is data-oriented and captures the distribution of the underlying database to make cardinality predictions. In addition, there are also works for online query optimization, which perform data sampling and plan cost evaluation to identify the optimal plan during

query execution time.

#### 2.2.1.1 Supervised Cardinality and Cost Estimation

Supervised cardinality and cost estimation refer to the methods which take the input of (a large number of) $\langle Q, C \rangle$ pairs to train a model, where $Q$ denotes a query and $C$ denotes the (exact or approximated) cardinality or cost of the query. Given a new query, the model directly predicts its cardinality or cost.

**Query encoding**. Queries in the form of SQL-like language need to be encoded so that the ML model can understand the semantics. A representative example of query encoding is provided in [85] where a query is encoded with a tuple (T,J,P), denoting the tables, join attributes, and predicate attributes and values involved in the query respectively. It is also observed [38] that integrating information such as the estimated selectivity of each table on a join query into the query encoding would benefit the overall cardinality estimation.

**Models**. Kipf et al. [85] use a multi-set convolutional model to map the query tuple into the corresponding cardinality. More specifically, a Multi-layer Perceptron (MLP) is trained for the table set, join attribute set, and predicate set respectively, and the output of the three MLPs are then feed into another network to predict the cardinality.

Sun et al. [152] propose a framework which directly estimates the cost (rather than cardinality) of a given query plan. The model designed in the paper consists of three components: (1) embedding layer, which converts the one-hot representations of metadata, operators, etc. into dense vectors to ease training; (2) representation layer, which is a tree-like network (just like a query plan) and recursively utilizes the representation of an operator or a sub-plan to output the representation of a more complete new plan, until the entire plan is embedded; (3) estimation layer, which takes the plan representation and output the corresponding cost.

Dutt et al. [38] focus on the scenario when training efficiency is a concern and observe that lightweight models such as XG-Boost achieve similar estimation accuracy as deep models but greatly reduces the training time and space overhead.

**Iterative refinement**. Dutt et al. [38] approximate the cardinality of a given query with uniformly selected tuples to generate the training data, avoiding the expensive cost of executing the query and obtaining the true cardinality. The authors design a method

which, given an error threshold, automatically computes the number of random tuples required to approximate the cardinality as well as the number of required training samples to satisfy the accuracy requirement. In general, lower error thresholds require more tuples for approximation and also incurs longer model training time.

Park et al. [129] propose methods to learn and refine the selectivity of a query on the fly. More specifically, the authors initially train a model off-line, and then iteratively refine the model when a query plan is executed and the true cardinality is observed.

#### 2.2.1.2 Unsupervised Cardinality Estimation

Unsupervised cardinality estimation approaches are agnostic to the query and workload, and learn the (joint or conditional) distribution of tuples in the database for cardinality estimation.

**Model**. The model frequently used by unsupervised cardinality estimation is autoregressive (AR) model, which has shown superior performance in density estimation for image and text data. In order to cope with the task of cardinality estimation, AR model is used to learn the conditional density distribution between attributes $A_1, A_2, \cdots, A_m$, e.g., the probability of attribute $A_2 = a_2$ given $A_1 = a_1$. An AR model is trained for each attribute, and the functionality of the $i$-th model is to to capture the conditional probability distribution of attribute $A_i$, i.e., $P(A_i|A_1, A_2, \cdots, A_{i-1})$. Given a query $A_1 = a_1, A_2 = a_2, \cdots, A_m = a_m$, the cardinality is computed as follows:

$$\prod_{i=1}^{m} P(A_i = a_i | A_1 = a_1 \cdots, A_{i-1} = a_{i-1}) \tag{2.1}$$

where $P(A_i = a_i|A_1 = a_1 \cdots, A_{i-1} = a_{i-1})$ is the output of the $i$-th network.

**Frameworks.** Yang et al. [187] design a framework named **Naru** with AR model being its core. As stated above, AR models are mainly used for point density estimation, and thus can only handle equality queries. Naru thus utilizes a Monte Carlo integration technique called progressive sampling to estimate the cardinality of range queries. More specifically, the AR models steer the sampler to regions with high density to reduce sampling cost, and importance sampling is then used to eliminate the induced bias.

**NeuroCard** [188] extends Naru to cardinality estimation for multi-table join queries. The authors train a model on the full outer join of all tables in the database which can thus handle join on any subsets of tables. Since the full outer join results in a huge amount of tuples, the authors propose an efficient sampling strategy which preserves the weights of all tuples, i.e., if a join key is more frequent in the join results, it is more frequent in the samples as well. The authors showcase that such sampling weights are fast to compute in practice, making the method applicable to large databases and complex queries.

Hasan et al. [62] study both using AR model for unsupervised cardinality estimation, and using $\langle Q, C \rangle$ pairs for supervised cardinality estimation. The authors also propose a method in dealing with database updates: training on the newly-inserted data using the original model configurations and weights for initialization, and adopting a dropout mechanism to prevent forgetting. The authors showcase that the model can adapt itself to different workloads by improving the weights of frequently queries attributes when training the model.

### 2.2.2 Integrating ML-based Query Optimizer into DBMSs

Researchers have also integrated ML-based optimizers into databases such as PostgreSQL for end-to-end query optimization to replace traditional optimizers and observe promising results.

A novel DBMS query processing engine based on Reinforcement Learning-based is proposed in SkinnerDB [163], which maintains no information regarding the query or the database. It partitions the execution of a query into time slices and tries different join plans at different slices using Reinforcement Learning techniques to find and finally stick to the optimal plan. A special query engine is designed to support slice-based query execution. Other works also show promising results in utilizing RL for join order selection, each with a special focus on different aspects in the selection process [91, 113, 190].

**Neo** [117] is an end-to-end query optimizer. It uses a value network for cost estimation, and a best-first greedy strategy for sub-plan growth to find the optimal plan. The training of Neo involves two phases: first, it uses ⟨Plan,Cost⟩ pairs generated by a second optimizer (e.g., the default optimizer of PostgreSQL) to train the value network from scratch (note here the Cost is approximated), and then use the true cost of different plans observed at

query time to refine the value network for more accurate estimation.

**Bao** [115] is designed to assist traditional optimizers with hints for the optimal plan selection. Hints, or hard-wired rules for plan selection, are widely used in many commercial databases such as DB2 and SQL Server. Given a set of hints, Bao partitions the hints into subsets, and learns which subset(s) of hints may finally result in better performance for a given query. During plan selection time, Bao routes the underlying optimizer to the subsets of hints which it believes benefits the selection of the optimal plan.

Ma et al. [109] focus on the setting where the data used to train the cardinality estimator diverges from the real database and workload. The authors use Active Learning (AL) to improve the performance of the model on the real workload. More specifically, among all unexecuted plans whose true costs are unknown, the authors select a subset of plans to execute to acquire extra knowledge regarding the workload and execution environment, which brings exploration cost but may potentially benefit the performance of the estimator.

### 2.2.3 Robust Query Optimizer

Instead of a single value, previous works [8] have shown that modelling the anticipated performance of a plan as a probability distribution over possible cost values helps to improve the predictability and robustness of the DBMS. More specifically, [8] proposes robust query optimizer, which maintains pre-computed samples from the underlying database. The actual cardinality of a given query $Q$ is assumed to follow a beta distribution with parameters obtained from the evaluation of $Q$ on the pre-computed samples. The cardinality distribution is then transformed into costs at a user-specified probability level. This approach proves the superiority of utilizing distribution for plan selection over a single cost value in providing more robust query performance. However, it suffers from the following limitations on its applicability: (1) The given query needs to be evaluated on the saved samples, which incurs extra overhead and delays the query answering process, and the delay becomes more severe for databases with complex schemas as a larger sample set need to be maintained; (2) The distribution is constructed for cardinality rather than execution time. As a result, the technique is not applicable to cost estimators without a cardinality estimation module such as modern Machine Learning-based estimators which directly map a plan to its execution

time; (3) The distribution is constructed using pre-computed samples, and the wisdom of the cost estimation module is not sufficiently utilized.

### 2.2.4 Conformal Technique

Instead of the techniques designed in robust query optimizer, we seek for different approaches to construct the distribution of plan's execution time, based on conformal techniques. Conformal techniques are a family of statistical tools that aim to quantify the uncertainty of regression predictors by outputting prediction intervals [87, 97, 137, 151, 157]. Although there are various types of conformal techniques that have different interval designs, they share a similar strategy by calibrating the regression predictor using given triplets of a sample, estimation, and ground truth. At a high level, given a regression predictor $\hat{\mu}$, conformal techniques take input of a collection of samples in the format of $(X, y, \hat{\mu}(X))$ (i.e., a feature vector, the response variable, and the estimated response variable), and fit a mapping function from $X$ to an interval $[l(X), u(X)]$ such that $y \in [l(X), u(X)]$ is true at a pre-defined probability level $1 - \alpha$. Representative conformal technique include (1) Split Conformal Technique [97], which computes the absolute estimation errors (residuals) $|\mu(\hat{X}) - y|$ for given samples and use these residuals to construct the intervals for new samples; and (2) Neural-network-based Prediction Intervals [86] which are designed especially for deep learning regression models and supplement the last layer of the network with two extra neurons estimating the $\frac{\alpha}{2}$-th and $1 - \frac{\alpha}{2}$-th quantile of the response variable respectively. The outputs of the two neurons form an interval which is expected to cover the response variable with probability $1-\alpha$. Conformal techniques make no assumption regarding the data distribution and thus are practical and robust.

### 2.2.5 Summary

The database community has seen promising results in applying ML techniques, including supervised learning, unsupervised learning, and Reinforcement Learning, for query optimization. Several practical problems in integrating ML techniques to database have also been considered. Many research chances to further improve the performance of ML-based query

optimizers still exist. For example, it is known that off-line model training cannot handle drifts in data and workload well, while online model retraining or refinement incurs extra time cost, and therefore a hybrid approach which can integrate the power of both methods, i.e., train an optimizer off-line and perform online on-demand model refinement, is expected to further benefit query optimization. Furthermore, current applications of ML-based query optimizers are mainly within laboratory databases for research exploration, and applying such techniques to real world databases designed for various scenarios (e.g., cloud-native database) is an interesting and important problem to tackle.

## 2.3 Data Market and Data Exchange

Data management methods and experiences have also accelerated the development of Machine Learning. Online data markets facilitate the access of training data with high quality [1, 10, 17, 20, 43, 103, 118, 132, 146]. And data cleaning, acquisition, augmentation techniques and the experience in dealing with relation data and large scale data broaden the application scenarios of ML models [2, 21, 22, 31, 42, 63, 88, 96, 133, 128, 142, 145, 192].

### 2.3.1 Data Market and Data Pricing

**Data market**. The industry has seen the rapid development of many online data markets such as Dawex [25], WorldQuant [175], and Xignite [183], which aim to make access to data a commodity, for modelling or learning purposes. In the academic area, Fernandez et al. [43] present their vision in the design of a data market for the trade of Machine Learning training data. The paper introduces the framework of a data market which involves data providers who own training data and are willing to trade the data for profits, data consumers who purchase data for various types of tasks such as training Machine Learning models, and arbiters who coordinate the trade between the provider and the consumer. The authors also provide a research agenda centering around the proper design of a data market and the rules of a healthy market such as arbitrage-free.

**Query-based pricing**. Chawla [17] study the problem of a consumer asking a query and paying a price for the query results. The authors design a pricing function which set a

price to a query according to the amount of revealed information with two considerations: (1) arbitrage-freeness, i.e., if the results of query $Q_i$ can be obtained from the results of query $Q_j$, then the price of answering $Q_j$ should be higher than answering $Q_i$; (2) revenue maximization, the data provider's revenue in selling the query results is maximized.

**Model-based pricing**. Chen et al. [20] study the problem of directly selling trained Machine Learning models on a data market. The authors adopt a method which first trains an optimal model on all available data, and for a given price offered by the data consumer, random Gaussian noise is injected to the model to prevent potential arbitrage opportunities. The authors demonstrate that the noise injection process incurs very small overhead and thus the technique can thus be applied on real-time online markets.

### 2.3.2 Data Acquisition and Data Augmentation

Data acquisition and augmentation techniques have also been adopted to acquire data (via combining old data) on demand to improve the performance of certain models. Data acquisition refers to the process to collect data (with a price) to finish a task, which is evaluated by a given metric, from a source such as a data seller. Data augmentation techniques find new features or feature combinations relevant to the user's tasks via operations such as table join.

**Data acquisition**. Chen et al. [21] consider the problem of obtaining data from multiple data providers in order to improve the accuracy of linear statistical estimators. The authors provide certain types of guarantees on the best strategies to adopt when providers decide to abstain from making their data available and thus data costs vary dynamically. In a related thread Kong et al. [88] study the problem of estimating Gaussian parameters and other estimators with Gaussian noise. Zhang et al. [192] study incentive mechanisms for participants in data markets to refresh their data.

**Data augmentation**. Data augmentation and feature selection for machine learning have also attracted research interests recently [22, 96, 145]. Kumar et al. [96] solve the problem of training a ML model on relational data and propose methods to decide whether joining certain attributes would improve the model's accuracy. Chepurko et al. [22] design an end-to-end Automatic Machine Learning (AML) system that automatically performs feature

selections and joins to generate new features that are expected to improve the performance of a given model.

### 2.3.3 Exploration vs. Exploitation

In Section 5 we consider a setting where a data consumer repetitively acquires data (with expense) from a data provider with a known objective. To maximize the gain (w.r.t the objective), the data consumer needs to solve the exploration and exploitation problem, and in this section we briefly introduce the related literature. Exploration refers to the process of obtaining new information regarding the task at some expense, and exploitation is to leverage the knowledge gained so far to select the optimal action. The trade-off between exploration and exploitation exists in many scenarios where a decision has to be made with incomplete knowledge.

**Thompson Sampling.** Methods balancing this trade-off have been proposed in various contexts, such as reinforcement learning [155] and online decision making [69], among which Thompson Sampling (TS) [139] is the most related work to the problem we study in Section 5. Given a set of actions $\mathcal{A}$ with each action $a \in \mathcal{A}$ bringing a reward $r$ drawn from a distribution $P_a(r)$, the objective of Thompson Sampling is to find a sequence of actions $a_1, a_2, \cdots, a_n$ such that the cumulative reward $r_1 + r_2 + \cdots + r_n$ is maximized. TS first makes assumptions regarding the reward distribution, i.e., $P_a(r)$, and updates $P_a(r)$ as more rewards of action $a$ are observed. Note that $P_a(r)$ may evolve over time and thus TS needs to continuously update its knowledge regarding $P_a(r)$. TS selects the next action according to its probability in maximizing the expected reward, balancing between exploration and exploitation. We give more detailed description of Thompson Sampling in Secton 5.4.1.

**Active Learning.** Active learning (AL) is another category of technique concerned with interactively acquiring labels for new data points to improve the performance of machine learning models [147]. It has been applied to solve various problems in data management [108]. In a typical setting for active learning, we have access to the features of new data and have to decide the set of records for which we would like to acquire the label for a cost. The most related work in this area to the problem studied in Section 5 is [105], which aims to control the class of data to acquire for improving the ML models. However, they focus

on the task of selecting class proportions for generating new training data. They do not explicitly optimize the use of a fixed budget, do not tackle regression problems, and require frequent re-training of the model.

### 2.3.4 Summary

Researchers in the data management community have developed frameworks such as online data market the corresponding data trade algorithms and principles to make ML model training data with high quality available and accelerate the wide deployment of ML techniques, and also proposed techniques regarding data cleaning and augmentation to further boost the performance of ML models. Current application of data management experience to ML area is still at the very early stage and many open questions need to be addressed. For example, current online data markets mainly tackle the data trade problem from the data provider's perspective, i.e., maximizing the provider's revenue in selling the data asset, while it is also very important to view the problem from the data consumer's perspective, such as maximizing the accuracy improvement of the consumer's model given a fixed budget.

# 3 LES$^3$: Learning-based Exact Set Similarity Search

## 3.1 Introduction

### 3.1.1 Background

Given a database $\mathcal{D}$ of sets each comprised of tokens (a token can be an arbitrary string from a given alphabet $\Sigma$, a unique identifier from a known domain, etc.), a single query set $Q$ (consisting of tokens from the same domain), and a similarity measure $Sim(*)$, the problem of *set similarity search* is to identify from $\mathcal{D}$ those sets that are within a user defined similarity threshold to the query $Q$ (range query) or $k$ sets that are the most similar to $Q$ ($k$NN query). This operation is essential to a wide spectrum of applications, such as data cleaning [57, 170], data integration [35, 51], query refinement [141], and digital trace analysis [101]. For example, a common task in data cleaning is to perform approximate string matching to identify near duplicates of a given query string. When strings are tokenized, the task of approximate string matching becomes a set similarity search problem. Given its prevalent use, efficient set similarity search is of paramount importance. A brute-force approach to supporting set similarity search is to scan all the sets in $\mathcal{D}$ and evaluate $Sim(*)$ between $Q$ and each set in $\mathcal{D}$ to obtain the results. When $\mathcal{D}$ is large or such operations are carried out repeatedly, however, its efficiency becomes a major concern.

Existing proposals to improve the search performance adopt a filter-and-verify framework: in the filter step, candidate sets are generated based on indexes on $\mathcal{D}$, and the candidate sets are further examined, computing the similarity between $Q$ and each candidate set in the verify step. Depending on the indexes used in the filter step, existing methods can be categorized into two groups: inverted index-based and tree-based. Inverted index-based methods build inverted index on tokens and only fetch those sets containing (a subset of)

tokens present in the query set as candidates. Tree-based methods [193, 194] transform sets to scalars [193] or vectors [194] and insert them into B+-trees or R-trees, which are then used at query processing time to quickly identify the candidate sets. As verification of a candidate set can be done very efficiently under almost all well-known set similarity measures (e.g., Jaccard, Dice, Cosine similarity) incurring a cost linear in the size of the set, optimization of the filter step is critical. Unfortunately, existing methods either utilize heavy-weight indexes that incur expensive storage consumption and excessive scanning cost during filtering [194], or employ indexes that are light-weight but with very limited pruning efficiency leading to an overly large candidate set [193]. Therefore, existing approaches mostly do not solve the set similarity search problem effectively. In fact for realistically low similarity thresholds or large result sizes, as we demonstrate in our experiments, the brute-force approach may perform much better.

### 3.1.2 Our Proposal

In this work, we study the problem of set similarity search, and propose a new approach named LES$^3$ (short for Learning-based Exact Set Similarity Search) that strives to reduce the time needed for filtering and increase the pruning efficiency of the index structure at the same time. At a high level, our approach also adopts a filter-and-verify framework; however we advocate the partitioning of the sets in $\mathcal{D}$ into non-overlapping *groups* for filtering. What differentiates our approach from existing methods is that instead of building complex index structures that could become too expensive to utilize at run-time, we introduce a light-weight index structure called *token-group matrix* (TGM); this structure is essentially a collection of bit-maps, to organize all groups, yielding comparable or higher pruning efficiency with only a fraction of storage cost and thus highly scalable. The TGM captures the association between tokens and groups, and allows us to quickly compute an upper bound on the similarity between the query set $Q$ and any set in a given group. Such upper bounds can then be used for *pruning* unrelated groups and directing search to the most promising groups.

As the search efficiency relies on the pruning efficiency of the TGM which in turn depends on how well the sets are partitioned, we formulate the construction of TGM as an optimization problem, that aims to identify the partitioning of sets that yields the highest

pruning efficiency. We first analytically model the base case in which every token has the same probability of appearing in any set. Our developments reveal that the optimal partitioning has two properties: balance and intra-group coherence. We then design a *general partitioning objective* ($GPO$) that strives to maximize the pruning efficiency, taking both properties into consideration.

We showcase that the optimal partitioning is NP-Hard and explore the use of algorithmic and machine learning-based methods to solve the optimization problem. Recent works [90, 49] have demonstrated that machine learning techniques have solid performance in learning the cumulative distribution function (CDF) of real data sets and this property can be used in important data management tasks such as indexing [99] and sorting [93]. We establish that machine learning techniques can also be utilized to produce superior solutions to hard optimization problems, central to other important indexing tasks such as those in support of set similarity search.

Complementary to existing works [49, 99] that utilize models such as piece-wise linear regression to learn a CDF, we explore models that are much better a fit, proposing a unique ensemble learning method suitable for progressive partitioning in our setting. The main difficulty in solving the optimization problem is that depending on $\mathcal{D}$, the number of groups needed for effective pruning can be large, so it is highly challenging to train a single network that would place any given set into one of these groups. As such, we propose a new learning framework named L2P (short for Learning to Partition) to address this challenge. L2P trains a cascade of Siamese networks to hierarchically partition the database $\mathcal{D}$ into increasingly finer groups until the desired number of groups is reached, resulting in $2^i$ groups at level $i$. The loss function for the Siamese network is specifically designed to minimize the distances between sets in the same group. As the input of a Siamese network has to be a vector, we devise a novel and efficient set representation method, *path-table representation* (PTR), that specifically caters to the needs of our optimization problem and proves to be a better fit than applicable embedding techniques. Although training ML models is known to be time-consuming, as will be shown in Section 5.5, L2P yields better partitioning results with much shorter processing time and only a small fraction of memory usage compared with other widely-adopted partitioning methods.

We fully develop the query processing algorithms for both range search and $k$NN search based on the TGM, and conduct extensive experiments on synthetic and real data sets to study the properties of our proposal and compare it against other applicable approaches. Our results demonstrate that both the proposed set representation method and the learning framework lead to much stronger pruning efficiency than competing methods. Overall, the proposed LES$^3$ method significantly outperforms the baseline methods in both memory-based and disk-based settings.

In summary, we make the following main contributions.

- We propose a learning-based approach, LES$^3$, for exact set similarity search, which partitions the database into groups to facilitate filtering. Central to LES$^3$ is TGM, a light-weight yet highly effective index that provides stronger pruning efficiency with less cost than state-of-the-art indexes.

- We formally analyze the partitioning of the database into groups, casting it as an optimization problem and discussing its distinction from well-studied clustering problems.

- We devise a novel learning framework, L2P, to solve the partitioning optimization problem, which yields significantly better partitioning results while incurring a small fraction of processing time and space cost compared with traditional algorithmic methods. L2P consists of a cascade of Siamese networks, an architecture that is able to effectively learn a partition of the dataset at different granularities, with up to thousands of groups at the finest level.

- We develop a carefully designed method for set representation, PTR, taking group separation into consideration. PTR theoretically and experimentally facilitates the training of L2P. Compared with other embedding techniques, PTR is orders of magnitude faster in computing set representations, and thus is more suitable for the target application where millions or billions of sets are involved (see Section 5.5).

- We experimentally study the performance of LES$^3$, L2P, and PTR, varying parameters of interest, including the network structure, number of groups and result size. We also examine the scalability of LES$^3$ utilizing real world large datasets in addition

to previously used set similarity benchmarks. The proposed methods significantly and consistently outperform competing methods across a large variety of settings, providing up to 5 times faster query processing and requiring up to 90% less space in typical scenarios.

## 3.2 Preliminaries

A *set* is an unordered collection of elements called *tokens* (we also consider multiset which may contain duplicate tokens). We use $S$ to denote an arbitrary set and $t$ an arbitrary token. The database $\mathcal{D}$ is a collection of sets, and all tokens form the token universe $\mathcal{T}$. Two sets are considered similar if the overlap in their tokens exceeds a user-defined threshold. Usually, such overlap is normalized to account for the size difference between sets. Examples of such similarity measures include Jaccard, Dice, and Cosine similarity. To make our discussion more concrete, we focus on Jaccard similarity, and discuss how our approach can be applied to other similarity measures in Section 3.3.2. Next we give the formal problem definitions.

**Definition 1.** ***kNN Search****. Given the database of sets $\mathcal{D}$, a set similarity measure $Sim(*)$, a query set[1] $Q$, and a result size $k$, find a collection $\mathcal{R}_Q^k \subseteq \mathcal{D}$ s.t. $|\mathcal{R}_Q^k| = k$ and $\forall S \in \mathcal{R}_Q^k$, $\forall S' \in \mathcal{D} - \mathcal{R}_Q^k$, $Sim(Q,S) \geq Sim(Q,S')$.*

**Definition 2.** ***Range Search****. Given the database of sets $\mathcal{D}$, a set similarity measure $Sim(*)$, a query set $Q$, and a threshold $\delta$, find a collection $\mathcal{R}_Q^\delta \subseteq \mathcal{D}$ s.t. $\forall S \in \mathcal{R}_Q^\delta$, $Sim(Q,S) \geq \delta$, and $\forall S' \in \mathcal{D} - \mathcal{R}_Q^\delta$, $Sim(Q,S') < \delta$.*

Our goal is to accelerate the process of identifying the result collection $\mathcal{R}_Q^k$ or $\mathcal{R}_Q^\delta$ for the given $k$ or $\delta$. In general, the query answering process consists of a filtering step (choosing candidate sets) and a verification step (comparing candidate sets with the query set). The cost of the verification step depends directly on the pruning efficiency of the search process, which measures the proportion of sets in $\mathcal{D}$ being pruned in the filtering step.

**Definition 3.** ***Pruning Efficiency (PE)****. Let $\mathcal{S}_Q$ be the collection of candidate sets for which the similarities to $Q$ must be computed in the process of identifying $\mathcal{R}_Q^k$ or $\mathcal{R}_Q^\delta$. Then*

[1]Without loss of generality, we assume that a query set consists of tokens existing in $\mathcal{T}$ only. The case of query sets containing tokens not in $\mathcal{T}$ can be handled similarly and is discussed in Section 3.3.1.

*the pruning efficiency of query processing, denoted as $PE$, is $\frac{|\mathcal{D}|-(|\mathcal{S}_Q|-k)}{|\mathcal{D}|}$ for kNN query, or $\frac{|\mathcal{D}|-(|\mathcal{S}_Q|-|\mathcal{R}_Q^\delta|)}{|\mathcal{D}|}$ for range query.*

Clearly $PE$ falls in the range $[0, 1]$. All other things being equal, a higher $PE$ leads to a lower verification cost. Our focus is therefore to design an approach for set similarity search that enjoys high PE and low filtering and verification cost.

## 3.3 Token-Group Matrix

The basic idea of our approach is to partition the sets in $\mathcal{D}$ into non-overlapping groups and index them properly, so that the search space can be pruned (i.e., certain groups can be quickly eliminated from further consideration) to speed up query processing. At the heart of our proposal is the token-group matrix (TGM), the index that records the relationship between tokens and the groups resulting from partitioning. In this section, we present the index structure and discuss its applicability across different similarity measures.

### 3.3.1 Index Structure

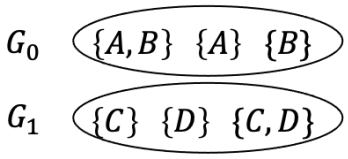
Assume for now, that $\mathcal{D}$ is already partitioned into $n$ non-overlapping groups, $\mathcal{G}_1, \cdots \mathcal{G}_n$; we defer the discussion of the strategies for partitioning to the next section. The goals of the index are simplicity (so that it incurs little computational and storage overhead) and effectiveness (providing high pruning efficiency). To this end, the TGM, $M$, with size $n * |\mathcal{T}|$, is constructed in the following way:

$$M[g,t] = \begin{cases} 1, & \text{if } \exists S \in \mathcal{G}_g \text{ s.t. } t \in S \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

where $t \in [1, |\mathcal{T}|]$ and $g \in [1, n]$.

An example of TGM is given in Figure 3.1, where $\mathcal{T} = \{A, B, C, D\}$ and six sets are partitioned into two groups $\mathcal{G}_0$ and $\mathcal{G}_1$.

The design of the TGM is based on the observation that when deciding whether a group of sets is a candidate for a query set or not, the only information needed is the number of common tokens they share. Such information can be easily obtained by visiting some




| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $G_0$ | 1 | 1 | 0 | 0 |
| $G_1$ | 0 | 0 | 1 | 1 |

Figure 3.1: An example of TGM

elements in $M$, and thus we can compute a *similarity upper bound* between a query set $Q$ and a group of sets $\mathcal{G}_g$, which are useful in pruning the search space, as follows:

$$UB(Q, \mathcal{G}_g) = \frac{\sum_{t \in Q} M[g, t]}{|Q|} \tag{3.2}$$

Continuing the example above, we assume that the query set is $\{A\}$ and $\mathcal{G}_0$ and $\mathcal{G}_1$ in Figure 3.1 are candidates. Then the similarity bound between the query set and $\mathcal{G}_0$ is $\frac{M[\mathcal{G}_0, A]}{|\{A\}|} = 1$, and the upper bound for $\mathcal{G}_1$ is $\frac{M[\mathcal{G}_1, A]}{|\{A\}|} = 0$.

Although we assume $Q$ contains tokens in $\mathcal{T}$ only, the case where this does not hold can be handled by letting $M[*, t'] = 0$ for $t' \notin \mathcal{T}$ in Equation (3.2). No further changes are required.

In the query processing step, if the upper bound of group $\mathcal{G}_g$ exceeds a threshold (can be $\delta$ in range query, or the minimal $k$NN similarity found so far), we compare all sets in $\mathcal{G}_g$ with the query set. The time complexity of computing the similarity bounds between the query set and all groups of sets is $O(n|Q|)$. It in general costs much less than computing the similarity between the query set and each set in $\mathcal{D}$, as the number of groups is usually orders of magnitude smaller than $|\mathcal{D}|$.

In terms of space consumption, each element in TGM is represented by a bit, and TGM is essentially a bitmap index. It is evident that $M$ is usually a very sparse matrix as each set usually contains a very small portion of the tokens from the universe. When necessary, many existing compression techniques [143, 144] can be employed to reduce the size of $M$.

### 3.3.2 Applicability

Although Equation (3.2) is computed assuming Jaccard index as the similarity metric, TGM works with many other set similarity measures as well, including measures that do not follow the triangle inequality, such as cosine similarity.

**Theorem 1.** *For $\forall Q, S \subseteq \mathcal{T}$, let $R = Q \cap S$. TGM is applicable to set similarity search tasks with measure $Sim(*)$ if*

1. *$Sim(Q, R) \geq Sim(Q, S)$, and*

2. *$\forall R' \subset R$, $Sim(Q, R) \geq Sim(Q, R')$*

*Proof.* We prove that for an arbitrary query set $Q$ and an arbitrary group $\mathcal{G}_g$, we can compute a similarity upper bound $UB(Q, \mathcal{G}_g)$ with TGM using Equation (3.2) such that $\forall S \in \mathcal{G}_g$, $UB(Q, \mathcal{G}_g) \geq Sim(Q, S)$. Let $R = \{t | t \in Q \wedge \exists S \in \mathcal{G}_g, t \in S\}$, then we know $\forall S \in \mathcal{G}_g$, $Q \cap S \subseteq R$. If $Q \cap S = R$, then clearly $Sim(Q, R) \geq Sim(Q, S)$; if $Q \cap S = R' \subset R$, then $Sim(Q, R) \geq Sim(Q, R') \geq Sim(Q, S)$. In either case, $Sim(Q, R)$ upper bounds $Sim(Q, S)$, and thus we can use $Sim(Q, R)$ as $UB(Q, \mathcal{G}_g)$. Since it is possible that $R = S$, in which case $Sim(Q, R) = Sim(Q, S)$, the bound $UB(Q, \mathcal{G}_g)$ is tight, even in multiset settings. □

For example, let $Q = \{t_1, t_2, t_3\}$ and $Q \cap S = \{t_1, t_2\}$. Then with Jaccard similarity, the set with the maximal similarity to $Q$ is $\{t_1, t_2\}$ and the upper bound is $\frac{2}{3}$; with cosine similarity, the set with the maximal similarity to $Q$ is also $\{t_1, t_2\}$, but the upper bound is $\frac{2}{\sqrt{3*2}} \approx 0.82$. Note that although most similarity measures satisfy the TGM Applicability Property, some exceptions do exist. One such example is the learned metric [84] which takes two samples (e.g., images) as the input and predicts their similarity.

In what follows, we call the two properties listed in Theorem 1 the *TGM Applicability Property*. Note that the token universe $\mathcal{T}$ does not need to be static. We will discuss how to adapt TGM to deal with cases where $\mathcal{T}$ is dynamically changing in Section 3.6.

## 3.4 Optimizing Partitioning

We analyze how to optimize partitioning to provide higher pruning efficiency. We discuss desired properties of the partitioning, and develop the objective function for the partitioning optimization problem that will guide the development of effective partitioning strategies. To make our formal analysis tractable, we make assumptions regarding the token distribution; nonetheless, as will be demonstrated by our experimental results in Section 5.5, the opti-

mization objectives and strategies thus developed are also expected to perform well when the assumptions do not hold.

### 3.4.1 The Case of Uniform Token Distribution

We formally analyze the effect of partitioning on pruning efficiency when the following assumption on token distribution holds.

**Definition 4.** ***Uniform Token Distribution Assumption****. The probabilities that different tokens belong to an arbitrary set are identical and independent. More specifically,* $\forall t_i, t_j \in \mathcal{T}$, $\forall S \in \mathcal{D}$, $P(t_i \in s) = P(t_j \in S)$, *and* $P(t_i \in S | t_j \in s) = P(t_i \in S | t_j \notin S)$.

For an arbitrary query $Q$, the expected pruning efficiency can be computed as follows:

$$E[PE] = \sum_{g=1}^{n} |\mathcal{G}_g|(1 - UB(Q, \mathcal{G}_g)) \tag{3.3}$$

Given the way the TGM is constructed, we rewrite Equation (3.2) in the following way to ease subsequent discussion:

$$UB(Q, \mathcal{G}_g) = \frac{\sum_{t \in Q} M[t, g]}{|Q|} = \frac{|GS_g \cap Q|}{|Q|},\ GS_g = \bigcup_{S \in \mathcal{G}_g} S \tag{3.4}$$

Accordingly, we rewrite Equation (3.3) as follows:

$$E[PE] = \sum_{g=1}^{n} |\mathcal{G}_g|(1 - \frac{|GS_g \cap Q|}{|Q|}) \tag{3.5}$$

As we assume $Q$ follows the same distribution as $\mathcal{D}$, $E[PE]$ over all possible $Q$ can be estimated by the following equation:

$$\frac{\sum_{Q \in \mathcal{D}} \sum_{g=1}^{n} |\mathcal{G}_g|(1 - \frac{|GS_g \cap Q|}{|Q|})}{|\mathcal{D}|} \tag{3.6}$$

Since $|\mathcal{D}|$ is a constant, we keep the nominator of Equation (3.6) only, and adjust the order as follows:

$$\sum_{g=1}^{n} |\mathcal{G}_g| \sum_{Q \in \mathcal{D}} (1 - \frac{|GS_g \cap Q|}{|Q|}) \tag{3.7}$$

To ease following analysis, we define term $F$ in Equation (3.8), and claim that maximizing

Equation (3.7) (and thus maximizing the pruning efficiency) is equivalent to minimizing $F$:

$$F = \sum_{g=1}^{n} |\mathcal{G}_g| \sum_{Q \in \mathcal{D}} \frac{|GS_g \cap Q|}{|Q|} \tag{3.8}$$

We derive several properties regarding the partitioning from Equation (3.8) so as to design practical partitioning algorithms.

**Theorem 2.** *In a database that satisfies the uniform token distribution assumption, the partitioning that minimizes Equation (3.8) produces groups with equal size (or differ by at most 1).*

*Proof.* We consider the special case where $\mathcal{D}$ is partitioned into two groups $\mathcal{G}_1$ and $\mathcal{G}_2$, and $|\mathcal{G}_1| \leq |\mathcal{G}_2|$. The $F$ value of such a partitioning is:

$$F = F_1 + F_2 = |\mathcal{G}_1| \sum_{Q \in \mathcal{D}} \frac{|GS_1 \cap Q|}{|Q|} + |\mathcal{G}_2| \sum_{Q \in \mathcal{D}} \frac{|GS_2 \cap Q|}{|Q|} \tag{3.9}$$

Next we move a set $S$ from $\mathcal{G}_1$ to $\mathcal{G}_2$ and prove that such movement increases the $F$ value. We know that if $S$ is moved from $\mathcal{G}_1$ to $\mathcal{G}_2$, $F_1$ would decrease and $F_2$ would increase. And since $|\mathcal{G}_1| \leq |\mathcal{G}_2|$, equivalently we can prove that $|\mathcal{G}_i| \sum_{Q \in \mathcal{D}} \frac{|GS_i \cap Q|}{|Q|}$ grows super-linearly with respect to $|\mathcal{G}_i|$, or $\sum_{Q \in \mathcal{D}} \frac{|GS_i \cap Q|}{|Q|}$ grows with $|\mathcal{G}_i|$. Given the construction of $GS_i$ in Equation (3.4), this is evidently true. Therefore, the $F$ value increases after the movement of $S$.

The above discussion can be naturally extended to multi-groups by moving one set from a small group to a large group each time, with the $F$ value increasing and the pruning efficiency decreasing during the process. In conclusion, balanced partitioning results yield the highest pruning efficiency. □

Even though the optimal partitioning is expected to produce groups with almost equal sizes, evidently balance is not the only desired property, according to Equation (3.8). We temporarily omit the $|\mathcal{G}_g|$ in Equation (3.8) and discuss other properties the partitioning must satisfy in order to provide higher pruning efficiency.

**Theorem 3.** *In a database that satisfies the uniform token distribution assumption, the partitioning that minimizes the following objective provides the highest pruning efficiency:*

$$\sum_{g=1}^{n} |\bigcup_{S \in \mathcal{G}_g} S| \tag{3.10}$$

*Proof.* Given the assumption that all groups are balanced, minimizing Equation (3.8) is equivalent to minimizing

$$\sum_{g=1}^{n} \sum_{Q \in \mathcal{D}} \frac{|GS_g \cap Q|}{|Q|} \tag{3.11}$$

Since $Q$ follows the uniform token distribution as well, which means that all tokens appear in $Q$ with the same probability, Equation (3.11) is proportional to the following equation:

$$\sum_{g=1}^{n} |GS_g \cap \mathcal{T}| = \sum_{g=1}^{n} |GS_g| = \sum_{g=1}^{n} |\bigcup_{S \in \mathcal{G}_g} S|, \tag{3.12}$$

where $\mathcal{T}$ denotes the token universe.

Thus, we can maximize PE by minimizing Equation (3.10). □

In summary, we have the following two desired properties regarding the partitioning of database $\mathcal{D}$.

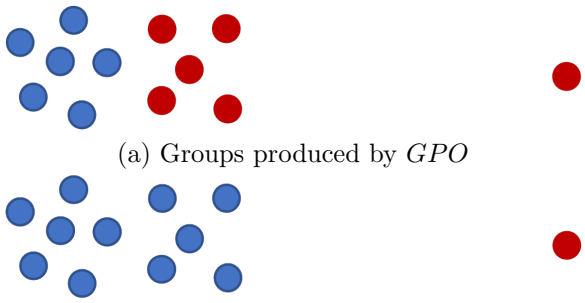
- Property 1: Groups are balanced;
- Property 2: $U = \sum_{g=1}^{n} |\bigcup_{S \in \mathcal{G}_g} S|$ is minimized.

### 3.4.2 The General Case

The analysis in the preceding section depends on the uniform token distribution assumption. In real-life datasets, this assumption does not hold. However, following the same methodology to derive a formal treatment of an arbitrary set/token distribution would be challenging as a realistic mathematical model of arbitrary set/token distributions would be hard to justify. Although the two properties identified above may not be true for optimal partitioning in the general case, we draw inspirations from them and propose a heuristic objective function that strives to maximize PE.

In essence Property 2 directs that the more similar (in terms of token composition) the sets are within a group, the better. We thus design a *general partitioning objective (GPO)* we wish to minimize reflecting this property:

$$GPO = \sum_{g=1}^{n} \sum_{S_x \in \mathcal{G}_g} \sum_{S_y \in \mathcal{G}_g} (1 - Sim(S_x, S_y)), \tag{3.13}$$



(a) Groups produced by $GPO$

(b) Groups produced by $k$-medians

Figure 3.2: Comparison of different partitioning results

where $Sim(*)$ can be any measures discussed in Section 3.3.2.

Intuitively, $GPO$ aims to minimize the sum of the intra-group pair-wise distances, where *distance* is defined as $1 - Sim(*)$. This is similar to Property 2. As an example, consider two groups $\mathcal{G}_i$ and $\mathcal{G}_j$ with $|\mathcal{G}_i| = |\mathcal{G}_j|$. Assume that $Sim(*)$ is Jaccard similarity, and we are to place a new set $S$ into one of the two groups. Then, if $\sum_{S_i \in \mathcal{G}_i}(1 - Sim(S, S_i)) < \sum_{S_j \in \mathcal{G}_j}(1 - Sim(S, S_j))$, that would mean $S$ shares more common tokens with sets in group $\mathcal{G}_i$, and thus inserting $S$ into group $\mathcal{G}_i$ helps to minimize $U$.

However, considering only Property 2 results in highly skewed partitioning results, as placing all sets in the same group provides the minimal $U$ (which equals to $|\mathcal{T}|$). Evidently, Property 1 is used to prevent such skewed partitioning in the uniform case. Luckily, $GPO$ enjoys a similar functionality: placing all sets in the same group provides $GPO = \sum_{S_x, S_y \in \mathcal{D}}(1 - Sim(S_x, S_y))$, which is the maximal possible $GPO$, and thus such a partitioning is never the optimal in terms of $GPO$. Thus, the design of $GPO$ implicitly incorporates both Property 1 and Property 2.

In order to better appreciate the distinctive value of the proposed partitioning objective, we compare $GPO$ with $k$-medians, perhaps the most popular clustering technique, and show by an example how optimizing $GPO$ leads to better results. We use a database of 21 sets, and the partitioning results based on different clustering objectives are given in Figure 3.2. Each set is represented by a point in the plot, and to better visualize the results we replace $(1 - Sim(*))$ with Euclidean distance.

Assume that the query is to identify the nearest neighbors of all 21 points. According to the search strategy of LES$^3$ given in Section 3.3.1, all points in the same group are candidates

of each other. Therefore, with the clustering results given in Figure 3.2(b), the total number of distance calculations is $20 * 20 + 1 * 1 = 401$, while with the partitioning results in Figure 3.2(a) the number is $13 * 13 + 8 * 8 = 233$. Clearly, the results based on Equation (3.13) have better pruning efficiency.

**Theorem 4.** *Given a database of sets $\mathcal{D}$ minimizing GPO on $\mathcal{D}$ is NP-complete.*

*Proof.* We give a brief proof by showing that minimizing *GPO* is essentially a 0-1 integer linear programming problem, which has been shown to be NP-complete [23]. More specifically, minimizing *GPO* is equivalent to solving the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \mathbf{e}_{|\mathcal{D}|} \cdot [\mathbf{A} \cdot \mathbf{A}^\intercal \odot \mathbf{D}] \cdot \mathbf{e}_{|\mathcal{D}|} \\ \text{subject to} \quad & \mathbf{e}_n \cdot \mathbf{A}^\intercal = \mathbf{e}_{|\mathcal{D}|} \end{aligned} \tag{3.14}$$

where $\mathbf{A}$ is a $|\mathcal{D}| \times n$ matrix and $\mathbf{A}[x, g] = 1$ if set $S_x$ belongs to group $\mathcal{G}_g$ and $\mathbf{A}[x, g] = 0$ otherwise, and $\mathbf{D}$ of size $|\mathcal{D}| \times |\mathcal{D}|$ denotes the distance matrix where $\mathbf{D}[x, y] = 1 - Sim(S_x, S_y)$, and $\mathbf{e}_i$ is a row vector of length $i$ filled with ones. The goal is to find the $\mathbf{A}$ which satisfies the constraint and maximizes the objective.

The intuition behind Equation (3.14) can be described as follows: $\mathbf{A} \cdot \mathbf{A}^\intercal$ is a $|\mathcal{D}| \times |\mathcal{D}|$ matrix such that the value at position $[x, y]$ is 1 if $S_x$ and $S_y$ belong to the same group, and 0 otherwise. The element-wise product between $\mathbf{A} \cdot \mathbf{A}^\intercal$ and $\mathbf{D}$ masks out those pair-wise distances between sets belonging to different groups, and $\mathbf{e}_{|\mathcal{D}|} \cdot [\mathbf{A} \cdot \mathbf{A}^\intercal \odot \mathbf{D}] \cdot \mathbf{e}_{|\mathcal{D}|}^\intercal$ sums the remaining distances, which is the same objective as *GPO*. The constraint $\mathbf{e}_n \cdot \mathbf{A}^\intercal = \mathbf{e}_{|\mathcal{D}|}$ guarantees that each set belongs to one and only one group. Therefore, minimizing *GPO* is equivalent to solving Equation (3.14), which completes the proof. □

### 3.4.3 Algorithmic Approaches

In this section we propose several algorithmic approaches based on existing applicable clustering methods, which are expected to yield groups with low *GPO* values. More specifically, we design a graph cut-based approach (PAR-G), a centroid-based approach (PAR-C), and a hierarchical approach (PAR-H).

#### 3.4.3.1 Graph cut-based method (PAR-G)

When $k$ or $\delta$ is fixed, it is possible to build an index structure specifically optimized for the workload. Dong et al. [36] propose a graph cut-based solution for (approximate) nearest neighbor search in $\mathbb{R}^d$ space by linking each point to its neighbors and partitioning the resulting graph into balanced subgraphs with the number of edges crossing different subgraphs minimized. Such a partitioning is shown to yield high pruning efficiency. Inspired by their approach, we design PAR-G, which takes $k$ or $\delta$ as one of its inputs, as follows:

1. **Similarity graph construction.** For a given $k$ in $k$NN query, construct the similarity graph, $G_{\mathcal{D}}$, of $\mathcal{D}$, such that $\forall S_x \in \mathcal{D}$, there exists a corresponding vertex $V_x$ in $G_{\mathcal{D}}$, and $\forall S_y \in \mathcal{D}$, if $S_y$ is one of the $k$ nearest neighbors of $S_x$, there is an edge between $V_x$ and $V_y$ in $G_{\mathcal{D}}$. For a given $\delta$ in range query, there is an edge between $V_x$ and $V_y$ if $Sim(S_x, S_y) \geq \delta$.

2. **Graph cut.** Partition $G_{\mathcal{D}}$ into $n$ balanced subgraphs while minimizing the number of edges crossing different subgraphs. This can be done with existing graph partitioners [79, 53].

#### 3.4.3.2 Centroid-based method (PAR-C)

Centroid-based methods [61] are iterative algorithms which at each iteration relocate an elements into a different cluster if such relocation improves the overall objective function. For our case, let $\phi(\mathcal{G}) = \sum_{S_x, S_y \in \mathcal{G}} (1 - Sim(S_x, S_y))$ be the sum of all pair-wise distances[2] in group $\mathcal{G}$, $S \in \mathcal{G}_i$ an arbitrary set, and $\Delta(S, \mathcal{G}_i, \mathcal{G}_j) = \phi(\mathcal{G}_i \setminus S) + \phi(\mathcal{G}_j \cup S) - \phi(\mathcal{G}_i) - \phi(\mathcal{G}_j)$ the decrease of $GPO$ after moving $S$ from $\mathcal{G}_i$ to $\mathcal{G}_j$ $(i, j \in [1, n])$. To be more specific, our method works as follows:

1. **Initialization.** Randomly partition $\mathcal{D}$ into $n$ groups;

2. **Relocation.** For each $S \in \mathcal{D}$, suppose $S \in \mathcal{G}_i$. Find group $\mathcal{G}_j^*$ such that $\Delta(S, \mathcal{G}_i, \mathcal{G}_j^*) = \max_{S, \mathcal{G}_i, \mathcal{G}_j} \Delta(S, \mathcal{G}_i, \mathcal{G}_j)$ (denoted as "the best group"). If $\Delta(S, \mathcal{G}_i, \mathcal{G}_j^*) > 0$, relocate $S$

[2]Note that repetitively calculating $\phi(\mathcal{G})$ during the partitioning process is computational prohibitive, and thus we approximate $\phi(\mathcal{G})$ with randomly selected sets in $\mathcal{G}$ in the experiment (Section 3.7.4).

from $\mathcal{G}_i$ to $\mathcal{G}_j^*$. Repeat this step until no sets are relocated in an iteration.

3. **Simplification.** Considering the data size we deal with (see Section 5.5), finding "the best group" at each iteration would be too expensive. Therefore, we adopt the "first-improvement" variant [159] of the algorithm, i.e., pick the first group $\mathcal{G}_j$ with $\Delta(S, \mathcal{G}_i, \mathcal{G}_j) > 0$ rather than the best group.

#### 3.4.3.3 Divisive clustering method (PAR-D)

Divisive clustering methods [80] start from the single cluster containing all elements and repeatedly split clusters until a desired number of clusters is reached. We reuse $\phi(\mathcal{G})$ introduced in Section 3.4.3.2 and use $idv\_d(S)$ to denote the sum of distances between $S$ and all other sets in the same group as $S$. PAR-D works as follows:

1. **Initialization.** Take $\mathcal{D}$ as the initial group.

2. **Splitting.** Find group $\mathcal{G}^* = \arg\max_{\mathcal{G}_i \in \{\mathcal{G}_1, \mathcal{G}_2, \cdots\}} \phi(\mathcal{G}_i)$, where $\{\mathcal{G}_1, \mathcal{G}_2, \cdots\}$ denotes all current groups. Find set $S^* = \arg\max_{S \in \mathcal{G}^*} idv\_d(S)$. Create a new group $\mathcal{G}^{new} = \{S^*\}$. For all other sets $S' \in \mathcal{G}^*$, move $S'$ to $\mathcal{G}^{new}$ if such movement reduces the overall $GPO$. Repeat this step until there are $n$ groups.

3. **Simplification.** Considering the data size we deal with, instead of finding $S^*$, we choose a random set in $\mathcal{G}^*$ to initialize $\mathcal{G}^{new}$, which is commonly adopted for group splitting [56].

#### 3.4.3.4 Agglomerative clustering method (PAR-A)

Agglomerative clustering [136] works in a bottom-up fashion by initially treating each element as a cluster and repeatedly merging clusters until a desired number of clusters is reached. We reuse $\phi(\mathcal{G})$ introduced in Section 3.4.3.2 to denote the sum of all pair-wise distances in group $\mathcal{G}$. PAR-A works as follows:

1. **Initialization.** Create group $\mathcal{G}_i = \{S_i\}$ for each $S_i \in \mathcal{D}$.

2. **Merging.** Find groups $\mathcal{G}_1^*, \mathcal{G}_2^* = \arg\min_{\mathcal{G}_i, \mathcal{G}_j \in \{\mathcal{G}_1, \mathcal{G}_2, \cdots\}}$ $\phi(\mathcal{G}_i \cup \mathcal{G}_j)$, where $\{\mathcal{G}_1, \mathcal{G}_2, \cdots\}$ denotes all current groups and $i \neq j$. Create a new group $\mathcal{G}^{new} = \mathcal{G}_1^* \cup \mathcal{G}_2^*$ and remove groups $\mathcal{G}_1^*$ and $\mathcal{G}_2^*$. Repeat this step until there are $n$ groups.

3. **Simplification.** Considering the data size we deal with, we adopt the heuristic that merging smaller groups (groups with smaller number of sets) usually results in smaller values of $\phi(\mathcal{G}_i \cup \mathcal{G}_j)$ and restrict that $\mathcal{G}_1^*$ is the smallest group (breaking ties randomly), and thus only $\mathcal{G}_2^*$ needs to be identified in each iteration.

As we will demonstrate in our experimental study in Section 7, these heuristic approaches do not provide satisfactory performance. The structure of the *GPO* problem objective does not resemble those targeted by well-studied clustering algorithms. In the next section, we explore the use of ML to perform such a partitioning.

## 3.5 L2P: Learn to Partition Sets into Groups

As pointed out by Bengio et al. [12], a machine learning approach to combinatorial optimization problems with well-defined objective functions, such as the Travelling Salesman Problem, has proven to be more promising than classical optimization methods with hand-wired rules in many scenarios, for the reason that it adapts solutions to the data and thus can uncover patterns in the specific problem instance as opposed to solving a general problem for every instance. It is widely agreed [166, 9] that ML-based methods are especially valuable in cases where expert knowledge of the problem domain may not be sufficient and some algorithmic decisions may not give satisfactory results. Our goal in this section, therefore, is to develop a machine learning method to optimize *GPO*.

### 3.5.1 Siamese Networks

Considering that the goal of optimizing *GPO* is to maximize the overall intra-group similarity, we adopt Siamese networks [15, 41] to solve the partitioning task. Siamese networks have been successfully utilized in deep metric learning tasks [122, 125] in computer vision,

capturing both intra-class similarity and inter-class discrimination in many challenging tasks including face recognition.

We design a Siamese network as shown in Figure 3.3 to learn the optimal partitioning. It consists of a pair of twin networks sharing the same set of parameters working in tandem on two inputs and generate two comparable outputs.

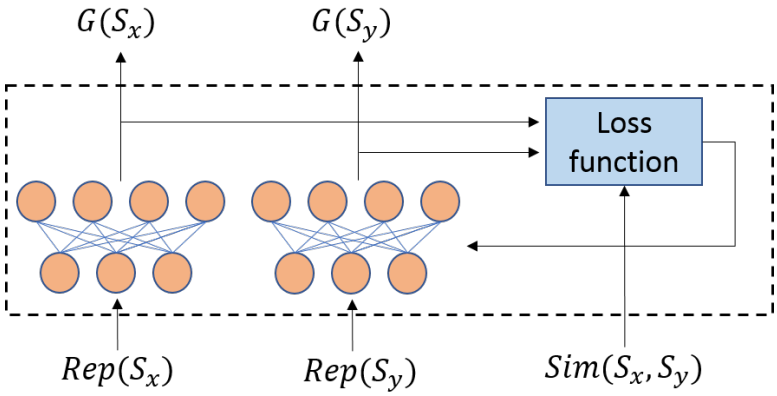



Figure 3.3: Siamese network

We use $Rep(S_x)$ and $Rep(S_y)$ to denote the vector representations of two sets $S_x$ and $S_y$, a pair of inputs to the twin networks, and use $G(S_x)$ and $G(S_y)$ to represent their respective group assignment indicted by the outputs of the twin networks respectively. Following Equation (3.13) we define the loss function of the Siamese network as follows:

$$loss(S_x, S_y) = \begin{cases} (1 - Sim(S_x, S_y)), & \text{if } G(S_x) = G(S_y) \\ 0, & \text{otherwise} \end{cases} \tag{3.15}$$

Equation (3.15) minimizes the intra-group dissimilarities by summing $(1 - sim(S_x, S_y))$ as the losses, and penalizes imbalanced groups by counting pairwise dissimilarities only between sets in the same group. We use an example to illustrate how Equation (3.15) penalizes imbalanced partitioning. Suppose there are $N$ sets and dissimilarity between any pair of sets is the same at $d$. The task is to partition these sets into 2 groups, containing $N_1$ and $N_2$ sets respectively ($N_1 + N_2 = N$). Then the overall loss is $\frac{N_1(N_1-1)d}{2} + \frac{N_2(N_2-1)d}{2} = \frac{d}{2}[N_1(N_1-1) + N_2(N_2-1)] = \frac{d}{2}(N_1^2 + N_2^2 - N) \geq \frac{d}{2}(\frac{(N_1+N_2)^2}{2} - N) = \frac{d}{2}(\frac{N^2}{2} - N)$, and the bound is tight when $N_1 = N_2$. Therefore, Equation (3.15) favors balanced partitioning.

By training the Siamese network with sufficient samples drawn from $\mathcal{D}$, theoretically we

can minimize the overall distances between all pairs of sets which belong to the same group, and thus the Siamese network is expected to give the partitioning result in which $GPO$ is minimized. Practically, however, we expect to achieve near-optimal partitioning only as the network is essentially performing local search.

### 3.5.2 Framework

Although using Siamese networks to solve the optimization problem is a promising approach, training such networks turns out to be difficult for the following reasons:

1. When dealing with real world data, we may need to partition sets into thousands of groups. Therefore, for an input set $S_x$, the network needs to make prediction on which group $S_x$ belongs to, among a collection of thousands of groups. It is well known [55] that training networks to tackle prediction problems involving thousands or more labels is challenging.

2. What makes this task even more difficult is that unlike a classification problem, the label for each input (i.e., the optimal group) in this optimization problem is unknown, i.e., there is no ground truth regarding the labels/groups available. The only information we have is the loss if the two input sets are assigned into the same group. This makes the problem even more challenging than typical classification problems.

The inherent difficulty of utilizing Siamese networks for this problem is the dimensionality (i.e., degrees of freedom) of the output space. In response to this challenge, we propose a learning framework consisting of a cascade of Siamese models, which partitions the database in a hierarchical fashion. Each Siamese network in the framework is responsible for partitioning a group of sets into two sub-groups. The framework is illustrated in Figure 3.4.

At Level 0 of the framework, we train a Siamese network which takes each set in the entire database $\mathcal{D}$ and assigns it into one of two groups, $\mathcal{G}_1$ and $\mathcal{G}_2$, based on the loss function given in Equation (3.15). Then at Level 1, we train two Siamese networks working on $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively in the same fashion. Thus, they partition the entire database into four groups. We continue adding more levels to the cascade framework until all groups are small enough

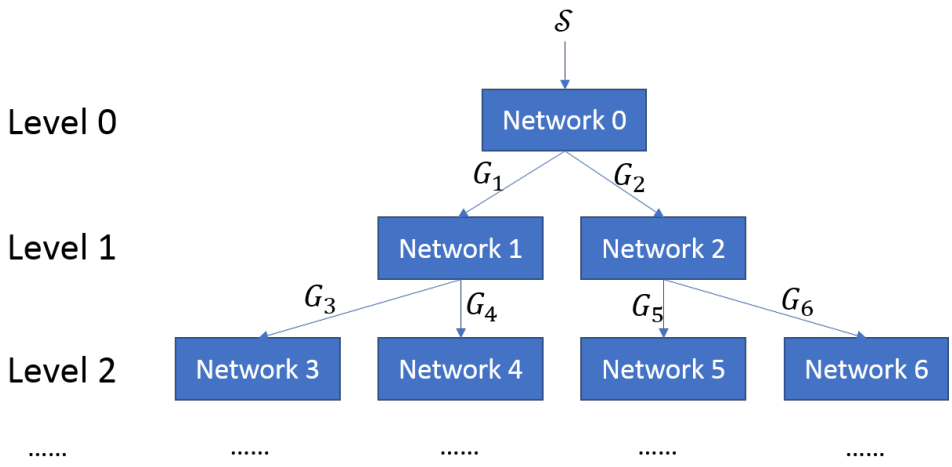



Figure 3.4: Cascade framework

or a pre-defined threshold on the number of levels is reached. Since each model in the cascade is specialized to partition a group of sets into only two sub-groups the resulting classification problem can be solved effectively.

The architecture of the cascade models motivates the use of a hierarchical indexing structure, which we call Hierarchical TGM (or HTGM). More specifically, assuming the level of the cascade framework is $l$, and $0 \leq i < j < l$, we construct $\text{TGM}_i$ and $\text{TGM}_j$ based on the partitioning results at level $i$ and level $j$ respectively. Suppose a group at level $i$, say $\mathcal{G}_g$, is partitioned into several sub-groups at level $j$, say $\mathcal{SG}_1, \cdots, \mathcal{SG}_m$. If for a query $Q$, group $\mathcal{G}_g$ can be pruned by checking $\text{TGM}_i$, then all verification operations involving groups $\mathcal{SG}_1, \cdots, \mathcal{SG}_m$ can be eliminated. The construction can be easily generalized to HTGM with $h$ ($h > 1$) levels.

### 3.5.3 PTR: a Set Representation Method

A Siamese network accepts vectors as input and thus the sets in $\mathcal{D}$ cannot be directly fed into the network. As a result we have to build a vector representation for each set. Considering the time and space complexity of existing embedding methods such as Principal Component Analysis (PCA) or Multidimensional Scaling (MDS), they can hardly be applied to the target setting introduced in Section 5.5 where millions or billions of sets are involved (see comparison regarding embedding cost in Section 3.7.3). Besides, different from the objectives of these

general-purpose embedding methods such as maximizing variance or preserving distance, our concern is to make sets containing different tokens more separable to benefit the training of the Siamese network. Intuitively, the representations that ease the training of the models are expected to bear the following property:

**Definition 5.** ***Set Separation-Friendly Property***. *$\forall t \in \mathcal{T}$, let $\mathcal{G}_t$ be the collection of sets containing $t$, and $\mathcal{G}_{\neg t}$ be the collection sets not containing $t$, then $\mathcal{G}_t$ and $\mathcal{G}_{\neg t}$ should to be easily separable in the representation space.*

Next we discuss how to construct such representations. As the first step, we organize all tokens with a balanced binary tree such that tokens appear in leaf nodes and each leaf contains only one token. The height of the tree is thus $h = \lceil \log_2 |\mathcal{T}| \rceil$. We mark the edge from a node to its left child with 1 and the edge to its right child with 0. An example of such a tree is depicted in Figure 3.5.

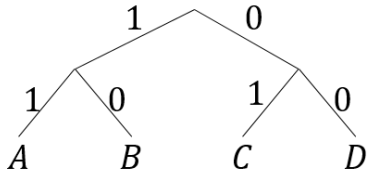



Figure 3.5: Tokens organized with a balanced tree

We use $path_t$ to denote the path from the root to an arbitrary token $t$. Since each leaf contains only one token, no two tokens share the same path. We build a path table (PT) of all tokens defined as follows:

$$\mathrm{PT}[t,i] = \begin{cases} path_t[i], & \text{if } i \in [1,h] \\ 1 - path_t[i-h], & \text{if } i \in [h+1, 2h] \end{cases} \tag{3.16}$$

An example of PT is provided in Table 3.1.

We propose a method called PTR (Path Table Representation) to build a representation for a given set $S$ as follows:

$$Rep(S)[i] = \sum_{t \in S} PT[t,i],\ i \in [1, 2h] \tag{3.17}$$

In the above example, the representation of $\{A, B, C\}$ is $[2, 2, 1, 1]$ and the representation of $\{B, D\}$ is $[1, 0, 1, 2]$. The second half of the path table (Positions 3 and 4) helps to reduce

Table 3.1: An example of path table (PT)

| Position | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 1 | 1 | 0 | 0 |
| B | 1 | 0 | 0 | 1 |
| C | 0 | 1 | 1 | 0 |
| D | 0 | 0 | 1 | 1 |

the chance that different sets have common representations. For example, if only the first half is used, then the representations of $\{A\}$, $\{B, C\}$, $\{A, D\}$, $\{B, C, D\}$ would all be $[1, 1]$. We compare the set representations constructed on the full vs. half path tables in Section 3.7.3.

PTR also naturally differentiates multisets containing the same collection of tokens but with different number of occurrences. For example, $Rep(\{A\}) = [1, 1, 0, 0]$ while $Rep(\{A, A\}) = [2, 2, 0, 0]$.

The basic idea of the representation is to map the sets into a new space, in a way that determining collections of sets containing specific tokens can be easily performed. More specifically, set $S$ is placed in the representation space based on the presence or absence of all tokens in $S$, and consequently, given a collection of tokens $\mathcal{T}_c$, we can quickly locate all sets containing $\mathcal{T}_c$. This evidently yields the set separation-friendly property. To better illustrate this, we reuse the path table in Table 3.1 and show that sets containing $B$ can be separated from other sets. For better visualization, we project the representation space onto the first two dimensions (Positions 1 and 2), and keep tokens $B$, $C$, and $D$ only in Figure 3.6. Clearly all sets containing token $B$ fall into the striped area, defined by the axis aligned hyper-plane in the representation space passing from point $(1, 0)$ (corresponding to $\{B\}$). Similarly all sets containing both $B$ and $C$ are located at the intersection of the axis aligned hyper-planes passing from $(1, 0)$ and $(0, 1)$ (corresponding to $\{C\}$). That way separating sets in the representation space based on token membership is conducted by determining hyper-plane intersections. We will demonstrate that such a representation is easier to learn and yields effective partitions in Section 5.5.

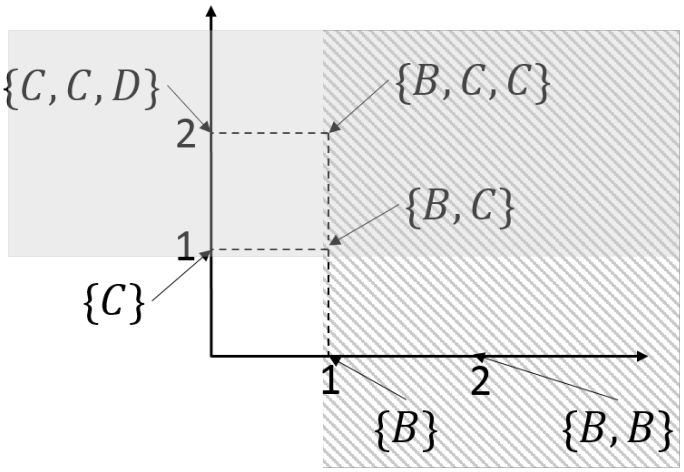



Figure 3.6: Separating sets

## 3.6 Dealing with Updates

Our discussions so far have assumed that the database $\mathcal{D}$ and the token universe are fixed. In some cases, however, new sets may be added to the database after the index is built, and previously unseen tokens may appear. We therefore study how updates can be handled, with a focus on TGM, as HTGM can be updated level by level in the same way.

We first discuss the case where new sets are added but the token universe remains the same. Given a new set $S$, we add $S$ into the group $\mathcal{G}_g$ if the similarity upper bound between $\mathcal{G}_g$ and $S$ is the highest among all groups. When there exist multiple groups with the same highest $UB$, we insert $S$ into the group with the minimal number of sets, in line with the optimization target discussed in Section 3.4. After insertion, we update the TGM accordingly, i.e., for all tokens $t \in S$, we set $M[g, t] = 1$.

We now demonstrate how our approach naturally handles previously unseen tokens. This is an important feature of our solution as it is the first to deal with dynamic tokens. All previous attempts to a solution of this problem assumed a fixed token universe [193, 194]. Let $S$ be a set containing one or more new tokens. We insert $S$ into the database in the following two steps:

1. Let $PS = S \cap \mathcal{T}$ be all tokens in $S$ that have been seen previously. We find the group with the highest similarity upper bound to $PS$, denoted by $\mathcal{G}_g$. If $PS = \emptyset$, then $\mathcal{G}_g$ is simply the group with the minimal number of sets. $S$ is inserted to $\mathcal{G}_g$.

2. For any token $t_{new} \in S \setminus PS$, add a row in $M$ corresponding to $t_{new}$. For all tokens $t \in S$, set $M[g, t] = 1$.

Although the partitioning in Section 3.4 is optimized based on existing sets and tokens, inserting new sets and tokens will not severely impact the performance of the approach, as we demonstrate in Section 3.7.8.

## 3.7 Experiments

In this section, we present a thorough experimental evaluation of our approach varying parameters of interest, comparing LES$^3$ and its important components, L2P and PTR, with competing methods.

### 3.7.1 Settings

**Environment.** We run the experiments on a machine with an Intel(R) Core i7-6700 CPU, 16GB memory and a 500GB, 5400 RPM HDD (roughly 80MB/s data read rate). We use HDD for fair comparison as other disk-based methods require no random access of the data (see Section 3.7.6). However one could expect better performance of LES$^3$ when running on SDD as it incurs random access of the data by skipping some groups, especially when the number of groups is large.

**Implementation.**[3] L2P is implemented with PyTorch, embedding methods in Section 3.7.3 and partitioning methods in Section 3.7.4 are implemented with Python, and TGM and the set similarity search baselines are implemented in C++ and compiled using GCC 9.3 with -O3 flag. TGM is compressed by Roaring [98], a well-performed bitmap compression technique.

**Datasets.** KOSARAK [89], LIVEJ [121], DBLP [26], and AOL [130] are three popular datasets used for set similarity search problems and we adapt them for this reason. We also include a social network dataset from Friendster[185] (denoted by FS), where each user is treated as a set with his/her friends being the tokens; and a dataset from PubMed Central

[3]code available at: https://github.com/AwesomeYifan/learning-based-set-sim-search

journal literature [76] (denoted by PMC), where each sentence is treated as a set with the words being the tokens[4]. Table 5.1 presents a summary of the statistics on these datasets. Considering the size of FS and PMC, we utilize them for disk-based evaluation in Section 3.7.6 to examine the scalability of LES[3].

Table 3.2: Dataset statistics

| Dataset | $\|\mathcal{D}\|$ | Set size | | | $\|\mathcal{T}\|$ |
|---|---|---|---|---|---|
| | | Max | Min | Avg | |
| KOSARAK | 990,002 | 2,498 | 1 | 8.1 | 41,270 |
| LIVEJ | 3,201,202 | 300 | 1 | 35.1 | 7,489,073 |
| DBLP | 5,875,251 | 462 | 2 | 8.7 | 3,720,067 |
| AOL | 10,154,742 | 245 | 1 | 3.0 | 3,849,555 |
| FS | 65,608,366 | 3,615 | 1 | 27.5 | 65,608,366 |
| PMC | 787,220,474 | 2,597 | 1 | 8.8 | 22,923,401 |

**Evaluation.** Following previous studies [28, 111, 194], we adopt Jaccard similarity as the metric in our experimental evaluation. We stress however that any similarity measures satisfying the TGM applicability property introduced in Section 3.3.2 can be adopted in our framework with highly similar results as those reported below. For each experiment, we randomly select 10K sets in the corresponding dataset as the queries and report the average search time. Unless otherwise specified, the indexing structure (TGM) and the data are memory-resident. We conduct disk-based evaluation in Section 3.7.6. We compare TGM with HTGM in Section 3.7.7. We select $n$ (number of groups) for each dataset which results in the shortest query latency. The influence of $n$ is studied in Section 3.7.5.

**Network and Loss Function.** We consider Multi-Layer Perceptron (two hidden layers, each consisting of eight neurons) and Sigmoid activation function for L2P training in the experiment and leave the investigation of other networks as future work. Clearly the network has one neuron at the output layer. Let $O_x$ be the output on input set $S_x$. If $O_x < 0.5$, then $S_x$ belongs to the first group; if $O_x \geq 0.5$, then $S_x$ belongs to the second group.

The loss function given in Equation (3.15) clearly describes the learning objective. However, it is difficult to train a network with the loss function as its gradient is 0. For efficient

[4]with basic data cleaning operations such as stop-words removal.

training, we use the following surrogate loss function, which leads to the same global optimum as Equation (3.15) while introducing non-zero gradients:

$$loss'(S_x, S_y) = \begin{cases} W(O_x, O_y)(1 - Sim(S_x, S_y)), & \text{if } V(O_x, O_y); \\ 0, & \text{otherwise;} \end{cases} \tag{3.18}$$

where $W(O_x, O_y) = (0.5 - |O_x - O_y|)$, and $V(O_x, O_y) = [(O_x \geq 0.5 \wedge O_y \geq 0.5) \vee (O_x < 0.5 \wedge O_y < 0.5)]$ indicating whether $S_x$ and $S_y$ are assigned to the same group ($V(O_x, O_y) = 1$) or not ($V(O_x, O_y) = 0$).

**Initialization.** Models at the first few levels of the Cascade framework deal with a large number of sets and incur long training time. To improve the training efficiency, we first sort all sets based on the minimal token contained in each set, and then partition all sets into 128 groups such that each group contains consecutive $|\mathcal{D}|/128$ sets, inspired by the idea of imposing sequential constraint to clustering tasks [156]. Since we always build TGM on the partitioning results at level 10 or higher which may contain thousands of groups, such initialization has minor impact on the performance but greatly reduces the training time. Note that initialization is not performed for the sampled dataset used in Section 3.7.3 due to its small size.

**Training.** For the Siamese network partitioning an arbitrary group, we randomly select 40,000 pairs of sets in the group to generate training samples, relatively small compared to the data size. It is observed that further increasing the number of training samples do not significantly improve the pruning efficiency of the partitioning results. We stop partitioning a group if it contains less than 50 sets, and thus the number of groups at level $i$ may be less than $2^i$. The batch size is set to 256, the number of epochs is set to 3 (except for Section 3.7.2 which reports the learning curves), and Adam is used as the optimizer. The same sampling-and-training procedure is repeated for each model in the cascade framework, starting from level 0.

### 3.7.2 Model Convergence and Training Cost

In this section we report the learning curves and the training costs. We observe that different models in the cascade framework introduced in Section 3.5.2 yield similar learning curves,

and thus we present the training loss of a random model at level 0 for each dataset (note that there are 128 models at level 0, see Section 4.5.1, paragraph Initialization). The training losses and costs are presented in Figure 3.7.

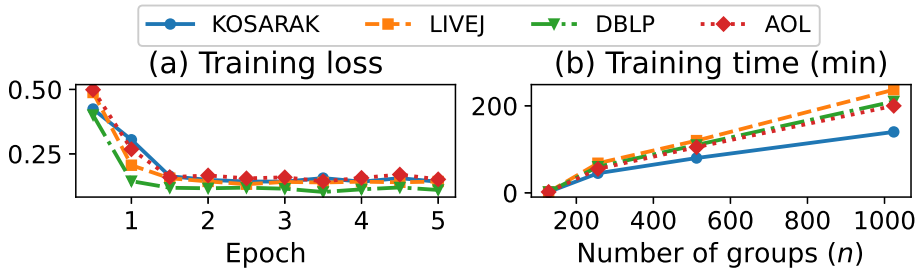



Figure 3.7: Training losses and costs

As is clear from Figure 3.7(a), on all datasets used in the experiment, the training loss decreases rapidly and the model converges after approximately two epochs, attesting to the efficiency of the model training process. Also, as can be observed from Figure 3.7(b), the training cost grows linearly with respect to the number of groups, making LES$^3$ scalable for large datasets. Besides, models at the same level of the cascade framework can be trained in parallel to further reduce the training cost, which is an interesting direction for future investigation.

### 3.7.3 PTR vs. Set Representation Techniques

We compare PTR with other applicable set representation techniques. More specifically, we choose PCA [74], a widely-used linear embedding method, MDS [27], a representative non-linear embedding approach, and Binary Encoding [58], an efficient categorical data embedding technique. We also include the variant of PTR constructed on the first half of the path table (see Section 3.5.3), denoted by PTR-half. Considering the complexity of PCA and MDS, we conduct experiments on sampled KOSARAK (sample ratio of 5%). We report the representation construction time of each method and the query answering time using the resulting partitioning results for $k$NN query ($k = 10$) and range query ($\delta = 0.7$) in Figure 3.8; similar trends are observed on other datasets and queries.

As can be observed from Figure 3.8, compared with PCA and MDS, PTR incurs much lower embedding time (10 to 20,000 times faster) while results in similar search time; com-

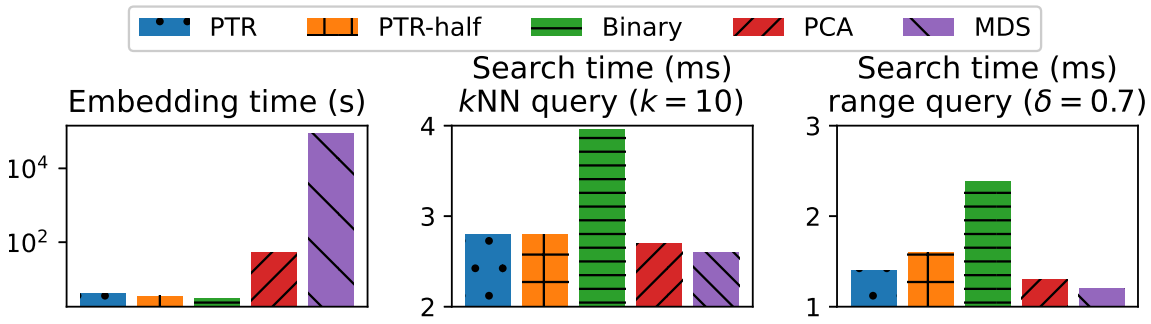



Figure 3.8: Comparison of representation techniques

pared with Binary Encoding and PTR-half, PTR leads to faster query answering with comparable embedding cost. Binary Encoding assigns unique representations to different sets without considering set characteristics (e.g., tokens contained therein), and thus can hardly achieve any Set Separation-Friendly Property. PTR-half, as discussed in Section 3.5.3, suffers from the risk that different sets may have common representations, and consequently these (dissimilar) sets are partitioned into the same group as they are not separable in the representation space, and the resulting search time thus is slightly longer than that of PTR. The major advantage of PTR is that it integrates the Set Separation-Friendly Property introduced in Section 3.5.3 into set representations by allowing sets consisting of different tokens to be easily separable by axis-aligned hyper-planes in the embedding space, and thus eases the training of the downstream Siamese networks.

### 3.7.4 L2P vs. Algorithmic Approaches

We compare the learning-based partitioning approach, L2P, to the algorithmic methods introduced in Section 3.4.3, namely the graph cut-based method (PAR-G), centroid-based method (PAR-C), divisive clustering method (PAR-D), and agglomerative clustering method (PAR-A), in terms of partitioning cost, including time cost and space cost, and query answering time.

For PAR-G, we adopt PaToH [16], a graph partitioning tool known to be efficient and performing well, to cut the graph. We report the cost of different methods in partitioning KOSARAK into 1024 groups and the query answering time for $k$NN with $k = 10$ in Figure 3.9; similar trends are observed on other datasets and queries. Note that the partitioning

time of L2P includes model training time and inference time (the time required to assign a set into a group), and the partitioning time of PAR-G consists of the $k$NN graph construction time and the graph cut time. PAR-G is specially optimized for $k = 10$ and the construction of its $k$NN graph is accelerated by LES$^3$.

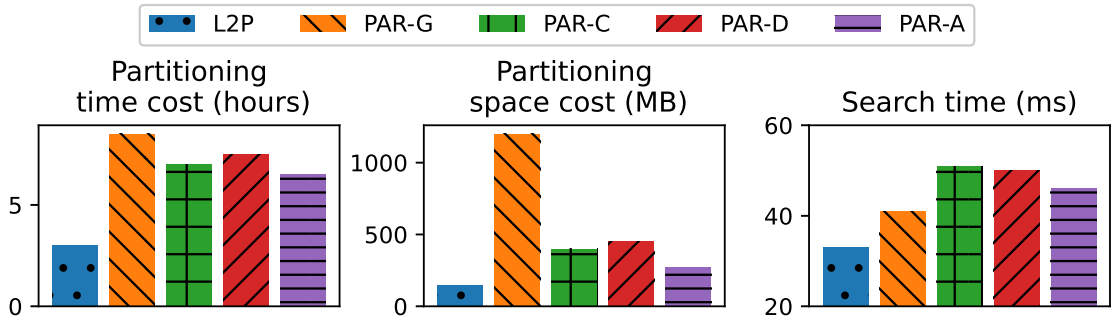



Figure 3.9: Comparison of partitioning methods

As depicted in Figure 3.9, L2P provides the fastest search while only incurs a small fraction of partitioning time and space cost compared to competitors (saving 80% partitioning time and 99% space compared with PAR-G). The reason why L2P incurs less partitioning time and space overhead is that, as described in Section 3.5.1 and Section 4.5.1, by training the models on a small portion of data, L2P is better positioned to approach the optimal partitioning where the $GPO$ is minimized, while other techniques work on the entire dataset and require (sometimes repetitively) computing the $GPO$ of arbitrary groups (or pairs) of sets. Besides, only model parameters and the training samples in a mini batch have to be saved in memory for L2P, with minimal storage overhead, while other techniques require materializing a large amount of intermediate partitioning results (and the entire $k$NN graph for PAR-G) in memory, incurring prohibitive space consumption.

By directly optimizing the $GPO$ which integrates the two desired properties of a partitioning with higher pruning efficiency, L2P is able to outperform PAR-G, the objective of which is minimizing the number of edges in the similarity graph crossing different sub-graphs rather than $GPO$. PAR-C, PAR-D, and PAR-A, although also aim to optimize the $GPO$, suffer from severe local optimality problems: a set is moved to a group only if such movement reduces the overall $GPO$, while in many cases movements temporarily increasing the $GPO$ must be allowed to determine a global optimum. For example, let $S_i \in \mathcal{G}_i$, $S_j \in \mathcal{G}_j$ be two

sets. Assume that moving $S_i$ to $\mathcal{G}_j$ and moving $S_j$ to $\mathcal{G}_i$ when individually carried out both increase the $GPO$, and consequently $S_i$ remains in $\mathcal{G}_j$ and $S_j$ in $\mathcal{G}_i$. However, swapping $S_i$ and $S_j$ may reduce the overall $GPO$ and thus leads to better partitioning. Such swapping cannot be achieved based on the strategy followed by PAR-C and PAR-D. Similarly, the strategy of PAR-A does not allow the merge of groups temporarily increasing the overall $GPO$, which however may be necessary in identifying a global optimum.

### 3.7.5 Sensitivity to Number of Groups and $k$

We test the performance of LES$^3$ in terms of query processing time on $k$NN queries, varying the number of groups $n$ and the result size $k$. The results are presented in Figure 3.10.

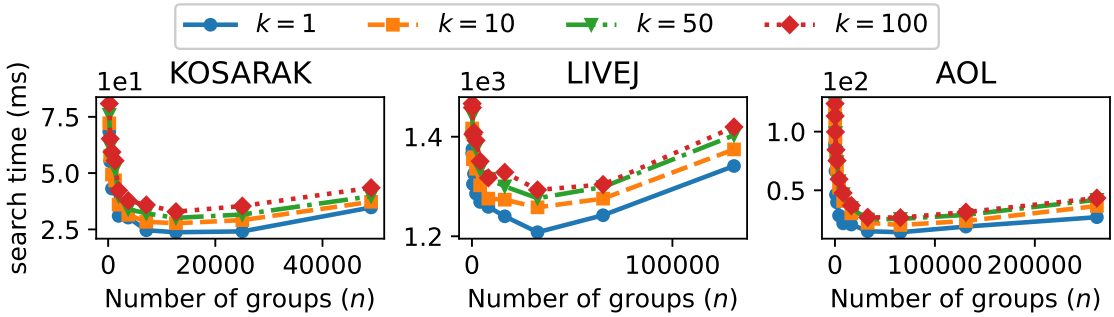



Figure 3.10: Sensitivity to the number of groups and result size

Increasing $n$ accelerates query answering, regardless of the result size. This is because with more groups, as indicated by Equation (3.13), the overall pruning efficiency of LES$^3$ can be improved, meaning fewer candidates have to be checked. Increasing $n$ benefits search time up to a point. In particular we observe a diminishing return behavior with respect to search performance as $n$ increases further. The reason is that, with a sufficiently large number of groups, sets are already well-separated, and further increasing $n$ brings no significant change to the pruning efficiency but incurs higher index (TGM) scan cost. Moreover, search time increases for larger $k$, which is consistent with our analysis in Section 3.4.1, as in general a larger $k$ in $k$NN search is analogous to a smaller $\delta$ in range search and thus the pruning efficiency is lower.

While determining the optimal number of groups for partitioning is a known NP-hard problem [71], we empirically observe from the experiments that setting the number of groups

at approximately $0.5\%|\mathcal{D}|$ leads to the lowest search time, where $|\mathcal{D}|$ is the number of sets in the corresponding dataset.

### 3.7.6 LES$^3$ vs. Set Similarity Search Baselines

In this section, we compare LES$^3$ with existing set similarity search approaches to answering $k$NN and range queries in memory-based and disk-based settings respectively. Among tree-based set similarity search approaches to date, DualTrans [194] provides the fastest query processing. For inverted index-based methods, we adopt the method proposed in [173] (denoted by InvIdx), which yields the state-of-the-art performance for set similarity join tasks. Note that we exclude methods requiring index construction during query time [28, 182, 29] as the index construction cost is much higher than the query cost. Since inverted index-based methods are designed for range queries and do not naturally support $k$NN queries, we modify the query answering algorithm of InvIdx for $k$NN queries as follows. (1) Given a query set $Q$ and a result size $k$, start with threshold $\delta = 1.0$ and use InvIdx to find candidate sets from $\mathcal{D}$ whose similarity with $Q$ exceeds $\delta$, denoted by $\mathcal{C}$. (2) Identify the temporary $k$NN results from $\mathcal{C}$, denoted by $\mathcal{R}_k$. If the minimal similarity between any set in $\mathcal{R}_k$ and $Q$ exceeds $\delta$, terminate. Otherwise, decrease $\delta$ by $z$, use InvIdx to find candidates sets with the new $\delta$, update $\mathcal{C}$ accordingly, and repeat the step. (3) Upon termination, $\mathcal{R}_k$ is guaranteed to be the $k$NN to $Q$, as the similarity between any sets in $\mathcal{D} \setminus \mathcal{C}$ and $Q$ does not exceed the current $\delta$. The value of $z$ is tuned for faster query answering.

In addition, we also include a brute-force approach, i.e., computing the similarity between the query set and all other sets to derive the results, for completeness of comparison.

In Figure 3.11, we show the index size and index construction time for all methods. It is clear that the indexing structure of LES$^3$, namely TGM, is much more lightweight, requiring up to 90% less space than DualTrans and InvIdx. The major time cost of constructing TGM comes from the model training, which however is a preprocessing step incurring only a one-time cost and can be further reduced as discussed in Section 3.7.2.

In Figure 3.12 we compare the performance of the four methods in a memory-based setting. We observe that LES$^3$ outperforms competitors for both $k$NN queries and range queries, accelerating the query answering by 2 to 20 times. DualTrans incurs longer search

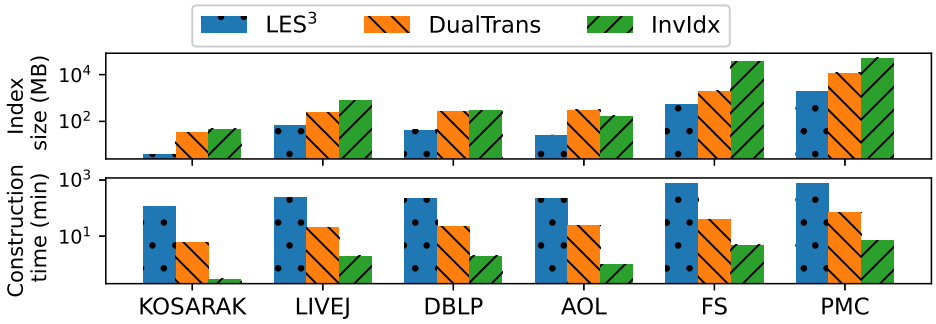



Figure 3.11: Index size and construction time

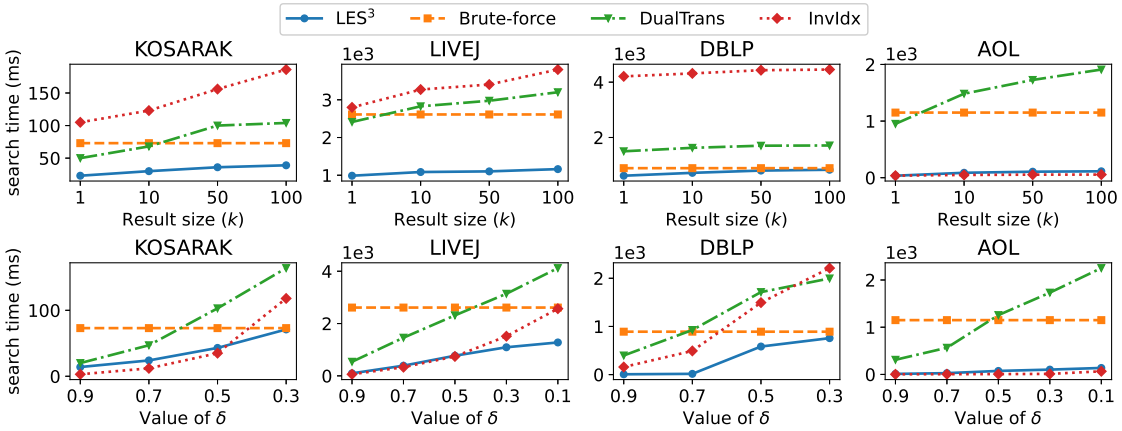



Figure 3.12: Comparison to baselines in memory-based settings for range queries (left) and $k$NN queries (right)

time as it uses an R-tree to organize all sets, with each set being represented with a $d$-dimensional vector ($d$ can be tuned for faster pruning). When the value of $d$ is small, sets containing different tokens cannot be clearly separated based on their representations, while when the value of $d$ is large, using R-tree to organize the vectors incurs high overlap between the bounding boxes of nodes on the R-tree, as previous research indicates [70]. Besides, scanning the R-tree is expensive, which is not worthwhile considering that set similarity (e.g., Jaccard similarity) can usually be computed efficiently. While InvIdx provides comparable performance with LES$^3$ for range queries with large $\delta$, it incurs greater search latency for $k$NN queries, especially when the average set size is large (e.g., on KOSARAK and LIVEJ). The reason is that, with InvIdx filtering operations need to be repeated for each candidate set (or multiple candidates with some common characteristics), and larger set size and $k$NN queries both enlarge the number of candidates, leading to sub-par query performance.

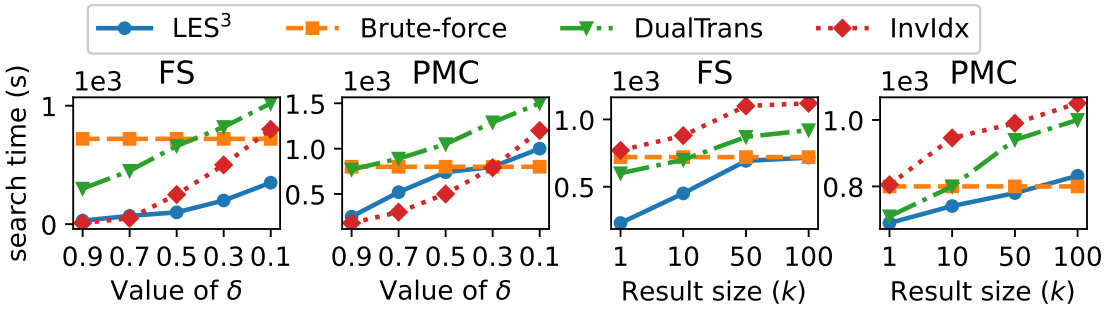



Figure 3.13: Comparison to baselines in disk-based settings for range queries (left) and $k$NN queries (right)

In contrast, we use TGM to compute the upper bounds between a query set and a group of sets; obtaining all bounds requires only $O(|S| * |\mathcal{G}|)$ time, which is relatively cheap. Although the search time of LES$^3$ increases for range query as $\delta$ decreases, LES$^3$ provide much faster query answering under a wide range of $\delta$.

We compare the performance of the four methods in the disk-based setting in Figure 3.13. Note that for DualTrans and InvIdx, only the part of the index that is necessary to the query answering, such as R-nodes on the search path and inverted indexes related to the query set, is retrieved into memory to reduce I/O cost. We observe that LES$^3$ generally provides faster search compared with competitors, accelerating the query answering by 2 to 10 times. The reasons why LES$^3$ incurs lower search time are: (1) Sets sharing no or very few common tokens with the query set can be easily pruned without being retrieved into memory; and (2) Since sets in the same group are checked jointly during the searching process; materializing a group of sets continuously on disk minimizes the data transfer delay. DualTrans and InvIdx, on the contrary, incur longer search latency and are outperformed by the Brute-force method for a wide range of $k$ and $\delta$. Besides the drawbacks discussed above in the memory-based setting, the search strategies of DualTrans and InvIdx incur repetitive retrieval of data with random disk access, which results in higher I/O cost (more pages retrieved, higher seek and rotation overhead, etc.), making them less efficient in the disk-based setting.

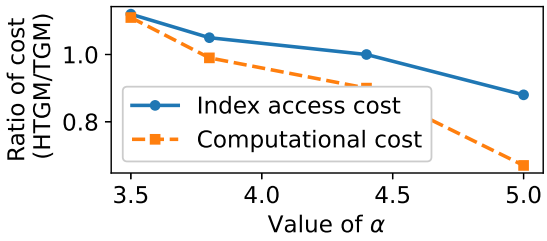



Figure 3.14: TGM vs. HTGM

### 3.7.7 TGM vs. HTGM

We evaluate the performance of TGM and HTGM to determine whether building a hierarchical index pays off. Intuitively, whether it benefits the query processing largely depends on the similarity distribution. For example, in cases where very few sets share common tokens, one can prune a large number of candidates using the matrices at the first few levels of HTGM, avoiding scanning the larger matrices at finer levels. However, in cases where most sets are similar, the small matrices at the first few levels of HTGM may provide no pruning efficiency at all. We assume that the similarity between sets in $\mathcal{D}$ can be modeled by a power-law distribution $P[sim = v] \sim v^{-\alpha}$, where $P[sim = v]$ denotes the probability that the similarity between any two sets is $v$, $v \in [0, 1]$, $\alpha \in [1, \infty)$. We generate multiple synthetic databases consisting of 20,000 sets and 20,000 tokens each, by varying the value of $\alpha$. We train a cascade model with 9 levels (including level 0). We use the partitioning results at level 8 (256 groups) to build the TGM, and use the partitioning results at level 5 (32 groups) and level 8 to build the HTGM. We compare HTGM and TGM from two aspects. First, the index access cost, measured by the number of columns in the HTGM or TGM that are checked when processing the query. Second, the computational cost, measured by the number of similarity calculations. We measure the ratio of cost between HTGM and TGM, and the results are shown in Figure 3.14. It is evident that HTGM outperforms TGM when the value of $\alpha$ is large, i.e., most sets are dissimilar. This is in line with the discussions in Section 3.7.7.

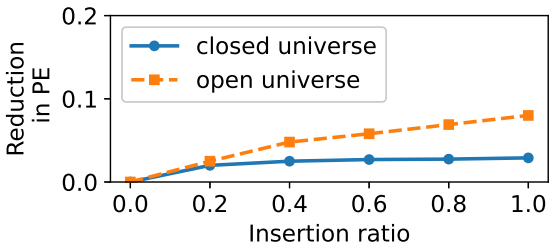



Figure 3.15: Handling updates

### 3.7.8 Handling Updates

We evaluate the performance of the proposed approach under updates. Two cases are considered: (1) *closed universe*, meaning the new sets to be inserted contain only tokens from the original database, and (2) *open universe*, where the new sets may contain previously unseen tokens. Let $\mathcal{D}$ be the original database, $\mathcal{D}^{closed}$ be the collection of new sets to be inserted under a closed universe, and $\mathcal{D}^{open}$ be the collection of new sets to be inserted under an open universes. For the experiment, we set insertion ratio ($|\mathcal{D}^{closed}|/|\mathcal{D}|$ and $|\mathcal{D}^{open}|/|\mathcal{D}|$) in range $[0, 1]$, and half of the tokens in $\mathcal{D}^{open}$ are from $\mathcal{D}$ and half are new. We compute the decrease in pruning efficiency after insertion compared to obtaining a partitioning from scratch (namely running L2P) on $\mathcal{D} \cup \mathcal{D}^{closed}$ or $\mathcal{D} \cup \mathcal{D}^{open}$ (referred to as *re-build*). We give the results for $k$NN query with $k = 10$ on KOSARAK in Figure 3.15; the experiments on the other datasets show similar trends.

Figure 3.15 depicts the percentage of pe reduction compared to re-build. The pruning efficiency decreases slightly as more new sets are inserted into the database. Insertions under an open universe have a higher impact on performance. The reason is that the tokens from the same universe mainly follow a similar distribution and the partition results obtained on the original data are still sufficient, while there is no prior knowledge regarding the distribution of new tokens. We observe, however, that the overall pruning efficiency is resistant to insertions (experiencing a decrease by at most 8%), which attests to the robustness of the proposed approach.

## 3.8 Summary

In this work, we have studied the problem of *exact set similarity search*, and designed LES$^3$, a filter-and-verify approach for efficient query processing. Central to our proposal is TGM, a simple yet effective structure that strikes a balance between index access cost and effectiveness in pruning candidate sets. We have revealed the desired properties of optimal partitioning in terms of pruning efficiency under the uniform token distribution assumption. We develop a learning-based approach, L2P, utilizing a cascade of Siamese networks to identify partitions. A novel set representation method, PTR, is developed to cater to the requirements of network training. The experimental results have demonstrated the superiority of LES$^3$ over other applicable approaches.

# 4 Execution Time Distribution-based Plan Selection

## 4.1 Introduction

### 4.1.1 Background

The predictable and controllable performance of Database Management System (DBMS) is of great importance across various scenarios [8, 162, 164] and especially when the DBMS serves as the core component of a larger system consisting of multiple applications. In practice the execution plan of a given query is selected based on an estimated cost which is expected to reveal the plan's potential behavior. However, due to data variance, limitation of the estimator, etc., using a single value to indicate the plan's behavior is usually inadequate and may cause sub-optimal performance.

Instead of a single value, previous works [8] have shown that modelling the cost of a plan as a probability distribution over possible values helps to improve the predictability and robustness of the DBMS. More specifically, the technique in [8] approximates the selectivity of a given query $Q$ based on pre-computed samples using a beta distribution, which is then transformed into a cost value at a desired probability level. This approach establishes the superiority of utilizing a distribution of costs for plan selection over a single cost value in terms of providing a more robust query performance. However, it suffers from the following limitations on its applicability: (1) The given query needs to be evaluated on the saved samples, which incurs extra overhead and delays the query answering process; the delay becomes more severe for databases with complex schemas as a larger sample set need to be maintained; (2) The distribution is constructed for selectivity rather than execution time (the time required to execute a plan). As a result, the technique is not applicable to cost estimators without a selectivity estimation module such as modern Machine Learning-based

estimator which directly maps a plan to its execution time.

### 4.1.2 Our Proposal

In this work, we propose a novel approach which leverages a black-box cost estimator (e.g., the default DBMS cost estimator or an external learned cost estimator [116, 153]) and directly produces the execution time distribution of plans. Our approach is based on a robust statistical technique of *conformal prediction* [87, 97, 137, 157]. At a high level, conformal prediction constructs a prediction interval around the output of a given regression predictor on an input $X$, and the interval is guaranteed to cover the actual output $y$ of the predictor at a predefined probability level $1 - \alpha$ (also referred to as calibration). Conformal predictions make no assumption regarding the joint distribution of $X$ and $y$, and is agnostic to the internal design of the regression predictor.

We consider execution time estimation as a prediction problem of the form $\hat{\mu} : X \rightarrow y$ where $X$ correspond to execution plans, $y$ are the execution times, and $\hat{\mu}$ is essentially an execution time estimator. Therefore, we can adapt conformal prediction techniques to calibrate the execution time estimator such that for a new plan $P$ and its estimated cost $\hat{\mu}(P)$, an interval around $\hat{\mu}(P)$ is constructed which covers the actual execution time of $P$ with probability at least $1 - \alpha$. Since conformal prediction produces an interval instead of a distribution, we design an intuitive and effective method to enhance it in the context of query optimization to produce the execution time distribution for a given plan. Readers are directed to Section 4.2.2 and Section 4.2.3 for the technical details of conformal prediction and the construction of the distribution.

In the past decades various methods have been proposed to predict the cost of query plans, such as selectivity estimator plus cost model as in standard DBMS, and learned cost estimator which leverage Machine Learning models to map plans to its execution times. The property of conformal prediction that treats the estimator as a black-box facilitates the construction of $\hat{\mu}$ from any of the existing cost estimators with any internal structure, fully leveraging the wisdom integrated in their designs and reducing extra overhead in inferring plan costs. We particularly study calibrating a conventional DBMS cost estimator and modern learned estimator for constructing the execution time distribution, proving its

effectiveness in improving the performance of these two important types of cost estimators.

The execution time distribution provides more comprehensive information regarding a plan's potential behavior and allows us to cater to different requirements arising from varying application scenarios. We specifically conduct case studies on three representative objectives, such as setting a threshold $\tau$ for a query $Q$ denoting that it is desired to get the results of $Q$ within time $\tau$, and setting a threshold $\tau$ and percentage $p$ for a query batch $\mathcal{Q}$ denoting that at least $p$ queries in $\mathcal{Q}$ need to be finished within $\tau$. We explore how the execution time distribution generated by conformal predictions can be utilized to identify suitable plans that best satisfy the given objectives.

To make the proposed execution time distribution-based plan selection strategies practical, we design an approach to interact with the DBMS using query hints, a feature generally supported by most DBMS (e.g., PostgreSQL, My SQL, SQL Server). Similar approaches have also been utilized in previous studies [116]. Our interaction method requires no change to the DBMS and only minor change to the query processing using query hints. More specifically, we implement the proposed execution time construction method and plan selection strategy as an external module to the DBMS. We leverage the DBMS' functionality to produce multiple candidate plans for a given query $Q$, and invoke this external module to identify the plan that best satisfies a user-specified objective, say $P^*$. As the last step, we interact with the DBMS by providing it related hints so that the DBMS would choose $P^*$ as the execution plan. As will be shown in Section 4.4, the integration of our proposal to DBMS is very lightweight, imposing only minor overheads to query processing, proving its effectiveness for practical deployment.

Our main contributions in this paper can be summarized as follows:

- We propose a novel approach to construct plan execution time distribution, which is assumption-free and can be applied to various types of plan cost estimators, making it a robust and highly generalizable technique to improve query performance.

- We design multiple intuitive and fundamental plan execution objectives based on the execution time distribution and the corresponding plan selection strategies, which help to satisfy the DBMS users' various requirement regarding the query performance.

- We design a lightweight approach to integrate our proposal into an existing DBMS which requires no change the the DBMS and only minor modification of the query processing procedure.

- We conduct extensive experiments on three benchmarks and with both conventional cost estimators and learned cost estimators. The results indicate that the execution time distribution helps to significantly improve the query performance in achieving each of the designed objective.

## 4.2 Execution Time Distribution

In this section we first introduce a statistical technique, called *conformal prediction* [157, 87, 138] which is originally introduced in the context of quantifying the uncertainty of regression predictors by wrapping the predictor's output with an interval which covers the actual response variable with predefined probability. We enhance conformal predictions to output probability distributions rather than intervals and adapt them to the problem studied herein to construct execution time distributions for query plan execution.

### 4.2.1 Conformal Predictions

Conformal predictions are a family of statistical tools that work on top of a regression predictor and construct prediction intervals based on the predictor's output. Such intervals attain coverage (i.e., the interval constructed for a given sample covering the actual value of the response variable) at a predefined probability level [137]. We introduce how conformal predictions work and the coverage guarantee they provide in this section.

Let $\hat{\mu} : X \rightarrow y$ be the regression predictor on which we apply conformal prediction, and $\{(X_i, y_i)\}_{i=1}^{n}$ be a collection of data samples, where $X_i \in \mathbb{R}^k$ denotes a feature vector and $y_i \in \mathbb{R}$ denotes the response variable. The conformal prediction techniques are initialized using samples $\{(X_i, y_i)\}_{i=1}^{n}$ such that for a given new feature vector $X_{n+1}$ which is exchangeable [5]

[5]Exchangeability is an assumption regarding the joint distribution of $X$ and $y$ which is less strict than i.i.d, details can be found in [97]

with $\{(X_i, y_i)\}_{i=1}^n$, it produces an interval $C_{\hat{\mu}}(X_{n+1})$ with the following *coverage guarantee*:

$$P(y_{n+1} \in C_{\hat{\mu}}(X_{n+1})) \geq 1 - \alpha \tag{4.1}$$

where $y_{n+1}$ is the response variable of $X_{n+1}$, $\alpha$ is a predefined probability level, and $C_{\hat{\mu}}(*)$ denotes the function to construct intervals for given inputs which is fit on samples $\{(X_i, y_i)\}_{i=1}^n$ during the offline initialization phase. Note that fitting function $C_{\hat{\mu}}(*)$ only requires the input $X_i$, the output $\hat{\mu}(X_i)$ of $\hat{\mu}$, and actual response variable $y_i$, and is agnostic to the internal process in $\hat{\mu}$ to map $X_i$ to $\hat{\mu}(X_i)$. The semantics of Equation (4.1) is that, the actual response variable of the given sample is covered by the constructed interval with a known probability.

In the initialization phase, various conformal prediction techniques may construct $C_{\hat{\mu}}(*)$ differently. For example, the technique in [97] takes each sample $(X_i, y_i) \in \{(X_i, y_i)\}_{i=1}^n$ and computes residual $R_i = |y_i - \hat{\mu}(X_i)|$, producing residuals $R_1, \cdots, R_n$. Assume that $d$ is the $\lceil n * (1 - \alpha) \rceil$-th smallest residual, then for the given new sample $X_{n+1}$, $[\hat{\mu}(X_{n+1}) - d, \hat{\mu}(X_{n+1}) + d]$ is used as the interval and Equation (4.1) is proven to hold [97].

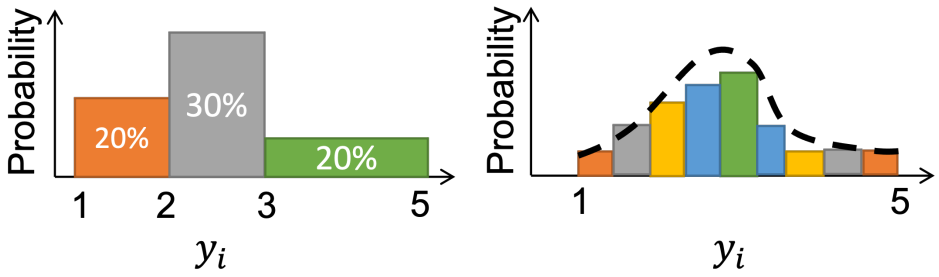
The value of $\alpha$ in Equation (4.1) is predefined by the user and reveals the probability that the intervals thus constructed covers the corresponding response variable (defined as *coverage rate*). For example, the intervals constructed for $X_{n+1}$ with $\alpha = 0.1$ covers $y_{n+1}$ with probability 90%. Intervals constructed with smaller $\alpha$ are expected to be wider since they have higher coverage rates.

Note that the coverage rate of conformal prediction is marginal instead of conditional [97, 137]. The difference is that, with marginal coverage, Equation (4.1) is true on average over randomly drawn sample $(X_{n+1}, y_{n+1})$, while with conditional coverage, Equation (4.1) holds for each individual sample $(X_{n+1}, y_{n+1})$. It is proven that conditional coverage is too strict to guarantee without impractical assumptions regarding the data distribution [48]. As will be shown in Section 4.2.3, marginal coverage is sufficient for the problem studied herein.

Compared with other applicable techniques for uncertainty quantification [8], conformal prediction makes no assumption regarding the data distribution, requires no change or minor change to the regression predictor $\hat{\mu}$, and incurs nearly negligible online overhead

Table 4.1: Discrete Distribution of $y_i$

| Probability level | interval |
|---|---|
| 70% | [1,5] |
| 30% | [2,3] |




(a) (b)

Figure 4.1: Execution Time Distribution

once function $C_{\hat{\mu}}(*)$ is obtained during the offline initialization phase. In addition, as will be shown in Section 4.2.3, conformal prediction treats $\hat{\mu}$ as a black-box and is agnostic to the implementation details of $\hat{\mu}$, which facilitates its applicability to various types of cost estimators.

### 4.2.2 Distribution Constructor

While with conformal prediction an interval is produced for a given input $X_i$ indicating the possible values of $y_i$, in practice it is usually desired to obtain a probability distribution over all possible values of $y_i$, rather than a single interval. In this section, we propose distribution constructor ($DC$), a method leveraging and enhancing conformal prediction to produce the distribution of $y_i$.

As shown in Equation (4.1), each interval is associated with a probability level $\alpha$, which is specified in advance. The intervals constructed with smaller $\alpha$ are wider than intervals associated with larger $\alpha$ as the former yield higher coverage rate. Using multiple $\alpha$s to construct intervals would thus produce multiple intervals with various width, forming a discrete distribution. In Table 4.1 and Figure 4.1, we provide an example of a distribution constructed using two different values of $\alpha$.

Note that the histogram in Figure 4.1(a) is transferred from the intervals in Table 4.1. The height of each bar is determined as follows: suppose that the probability of $y_i$ falling into interval $[l_1, u_1]$ is $1 - \alpha_1$, the probability of falling into interval $[l_2, u_2]$ is $1 - \alpha_2$, and $\alpha_1 < \alpha_2$, then the probabilities of falling in intervals $[l_1, l_2]$ or $[u_2, u_1]$ are both $\frac{\alpha_2 - \alpha_1}{2}$ [137].

The distribution constructor consists of multiple interval construction functions $C_{\hat{\mu}}$ introduced in Section 4.2.1 with different values of $\alpha$, and using more $\alpha$s gives more fine-grained distributions, producing the histogram (distribution) shown in Figure 4.1(b). We study the influence of the number of $\alpha$s on query performance in Section 4.5.7. Note that since conformal prediction provides marginal coverage, the execution time distribution thus constructed gives the same guarantee. More specifically, the probability associated with each bar in the distribution is marginal on average over random samples.

### 4.2.3 *DC* for Execution Time Distribution Construction

In this section we investigate how to build *DC* in the context of execution time distribution construction, so that it can be utilized for the planning and processing of incoming queries. It is clear from the description in Section 4.2.1 that there are two sub-tasks in constructing *DC*, namely building the regression predictor $\hat{\mu}$ and preparing samples $\{(X_i, y_i)\}_{i=1}^n$. For execution time distribution, $\hat{\mu}$ is evidently an execution time estimator, and $\{(X_i, y_i)\}_{i=1}^n$ are essentially (plan, execution time) pairs, which is referred to as $\{(P_i, t_i)\}_{i=1}^n$ in sequel. Note that besides execution time, the technique thus developed also applies to other metrics. For example, The user may be concerned about the CPU cost, in which case $\hat{\mu}$ is a CPU cost estimator and $\{(X_i, y_i)\}_{i=1}^n$ are (plan, CPU cost) pairs, and the distribution construction technique and plan selection objectives can naturally be reused.

We first look at the problem of building an execution time estimator. Plenty of methods have been proposed in the last decades to predict plan costs, such as cardinality estimator plus cost model (as in typical DBMS) [60, 123, 134, 179, 180] and Machine Learning-based cost estimator [78, 116, 150, 154, 176, 178]. These cost estimators have proven to be effective in various scenarios, and reusing them with necessary modification reduces the unnecessary labor and is thus highly desirable in our work.

Different cost estimators have varying internal designs. For example, conventional DBMS

cost estimators maintain auxiliary structures such as histograms to compute the cardinality of operators in a plan and adopt a cost model (consists of multiple equations) to derive the cost, and learned estimators train a Machine Learning (ML) model which takes the input of an encoded plan and outputs its anticipated execution time. Luckily, the property of conformal prediction that treats the regression predictor as a black-box, as stated in Section 4.2.1, makes it independent from the internal design and applicable to any type of estimators. As a result, our design in this work may be used together with any plan cost estimators, making it suitable for various scenarios where different estimators are preferred.

To construct an execution time estimator, necessary modifications to the cost estimator and the query processing process need to be involved. For example, a query plan in tree format must be encoded into vectors (e.g., using methods in [116, 153]) before feeding into the learned estimator; and the outputs of the conventional DBMS cost estimators are "costs", which are expected to be monotonic to but not equal execution times, and thus need to transformed into execution time using (non-)linear equations, ML models, etc. The process of constructing the execution time estimator is illustrated in Figure 4.2, where components in $\hat{\mu}$ are in the blue dashed box.

Note that in Figure 4.2, the specific functionality of the encoder and cost-execution time mapper depends on the type of estimator we use. For example, if learned estimators are used, the encoder performs necessary encoding to the plan, while for conventional cost estimators, this component needs not to be invoked; and the cost-execution time mapper can be a simple $f(x) = x$ function if the learned estimator is trained directly to predict execution times. The design in Figure 4.2, when integrated into a DBMS, incurs no change to the implementation of the cost estimator and only relies on its inputs and outputs, which reduces the overhead of applying the method to real systems.

Next we discuss the generation of samples $\{(P_i, t_i)\}_{i=1}^n$. We assume the existence of a query workload $\mathcal{Q}$, which can be the (subset of) queries provided together with benchmark datasets, or the history queries in a real DBMS. We then generate one or more candidate plan for each $Q \in \mathcal{Q}$ (we defer the discussion of the plan generation process to Section 4.4), and run each of the generated plan $P$ to get its execution time $t$, forming the set of samples $\{(P_i, t_i)\}_{i=1}^n$. Assuming all incoming queries following the same distribution as $\mathcal{Q}$

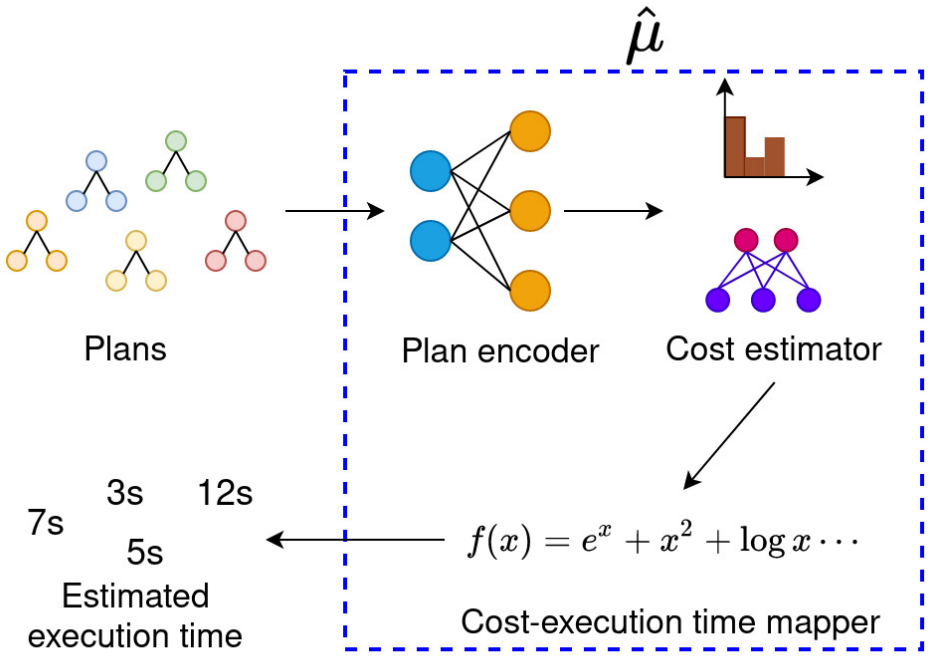



Figure 4.2: Constructing Execution Time Estimator

as in previous works [62, 177], and using the same process to generate candidate plans for incoming queries, the exchageability requirement of the conformal prediction is naturally satisfied.

For a given cost estimator *Estor* and a collection of samples $\{(P_i, t_i)\}_{i=1}^n$, the process of building *DC* can be summarized in Algorithm 1. Note that the interval construction function in line 6 of Algorithm 1 is introduced in Section 4.2.1. Recall that the distribution thus constructed provides marginal distribution on average over random plans. For the task of query optimization, the DBMS users are usually concerned about the overall performance of a workload (consists of many queries), and thus marginal coverage is sufficient.

### 4.2.4 Online Execution Time Distribution Construction

In this section we study how to utilize *DC* to construct execution time distribution with an imaginary plan $P_{new}$. We defer how $P_{new}$ is generated and how *DC* can be integrated into a real DBMS to Section 4.4. We present the process of producing the execution time

**Algorithm 1** Offline Preparation

**Input:** $Estor$, $\{(P_i, t_i)\}_{i=1}^{n}$, a set of $\alpha$ values $\mathcal{A}$
**Output:** Distribution constructor $DC$
1: Build encoder $Enc$ if necessary;
2: Build cost-execution time mapper $CEM$ if necessary;
3: Assemble $Estor$, $Enc$ and $CEM$ to build $\hat{\mu}$;
4: $DC = \emptyset$;
5: **for** each $\alpha \in \mathcal{A}$ **do**
6: Calibrate $\hat{\mu}$ using $\{(P_i, t_i)\}_{i=1}^{n}$ to obtain interval construction function $C_{\hat{\mu},\alpha}$;
7: $DC = DC \cup \{C_{\hat{\mu},\alpha}\}$;
8: **end for**
9: **return** $DC$;

distribution in Algorithm 2.

**Algorithm 2** Online Distribution Construction

**Input:** $DC$, plan $P_{new}$, a set of $\alpha$ values $\mathcal{A}$
**Output:** Execution time distribution $D$
1: $D = \emptyset$;
2: **for** each $\alpha \in \mathcal{A}$ **do**
3: Compute interval $I_\alpha = C_{\hat{\mu},\alpha}(P_{new})$;
4: $D = D \cup \{(\alpha, I_\alpha)\}$;
5: **end for**
6: **return** $D$;

We use the following example to illustrate how Algorithm 2 works and how the corresponding execution time distribution looks like.

**Example 4.2.1.** *Consider the plan in Figure 4.3 to join two tables $R$ and $T$ on attribute $a$.*

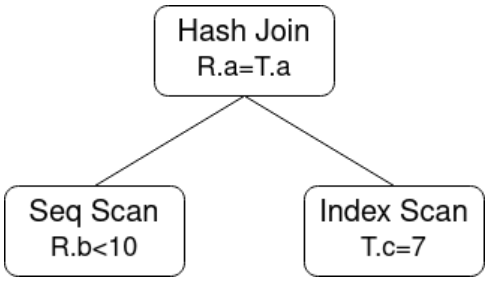



Figure 4.3: An Example Plan

*Assume that the distribution constructor $DC$ is constructed using $\alpha \in \{0.1, 0.3, 0.5\}$,*

*passing the plan to DC gives the execution time distribution in Table 4.2:*

Table 4.2: An Example Execution Time Distribution

| Value of $\alpha$ | Execution time interval |
|---|---|
| 0.1 | [1s,9s] |
| 0.3 | [3s,7s] |
| 0.5 | [4s,5s] |

*From the execution time distribution we know that the probability that the plan runs for over 7s is 15% ($\frac{1-0.3}{2}$), and the probability of running for over 5s is 25% ($\frac{1-0.5}{2}$), etc.*

Although the execution time distribution provides more information than a single estimated cost value, existing plan selection strategies only use simple cost comparison and thus cannot directly leverage such information. In addition, DBMS users in different scenarios may have various concerns regarding the plans' execution times and thus would use the distribution information differently. In the next section we design several fundamental and intuitive plan selection strategies based on execution time distribution.

## 4.3 Plan Selection Strategy

In Section 4.2 we have introduced how to build the execution time distribution for plans to provide more comprehensive description regarding the plan's potential behavior. In this section, we discuss how to use the execution time distribution as a guide to construct various intuitive and fundamental query execution objectives. More specifically, we consider two main categories of objectives, namely per-query objective and query batch objective, both related to the time budget in finishing the query/queries. We also design a plan selection strategy for each objective to maximize the probability that the objective can be satisfied. While there are many other ways to leverage the execution time distribution to benefit plan selection such as using different quantiles of the execution time distribution for risk-averse/risk-seeking plan selection as in [8], we focus on the two types of objectives mentioned above in this work as they are intuitive and of great interests to the DBMS user, and the investigation of other execution time distribution-based plan selection objectives and strategies is an interesting future work.

### 4.3.1 Per-query Execution time Threshold

In many cases, the user may have a time limit $\tau$ for a given query, indicating that the query results need to be returned within duration $\tau$. For example, an analyst may need the results to be back before a meeting starting in 10 minutes. We denote such constraint by *per-query execution time threshold*.

Under per-query execution time threshold, instead of selecting the plan with the lowest estimated cost, the user prefers the plan that has the highest probability to finish within the threshold. For example, as shown in Figure 4.4, while plan B has higher estimated execution time than plan A, its execution time distribution is narrow, and the probability that it runs for duration longer than $\tau$ (blue area) is smaller than that of plan A (blue area+orange area). Therefore, plan B is preferred when $\tau$ is the execution time threshold.

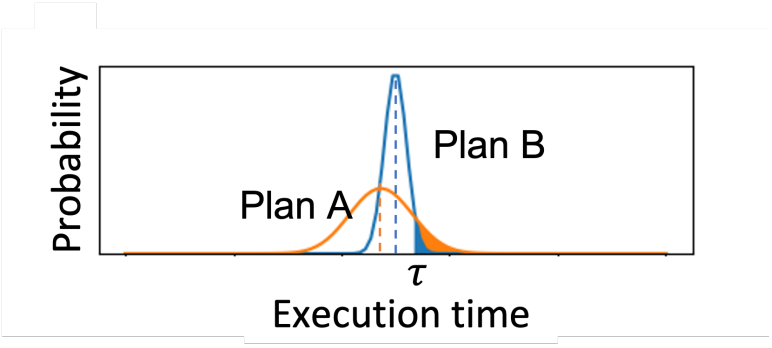



Figure 4.4: Execution Time Distributions of Two Plans

Given threshold $\tau$, based on the execution time distribution we compute the probability each plan can finish on time and select the one with the highest probability to satisfy such constraint. Note that there is no guarantee that the selected plan may finish within the time threshold, the only guarantee is that the selected plan has higher probability than other plans to satisfy the time constraint.

In order for the per-query execution time threshold to be practical and solvable, the user cannot specify an arbitrary value as the threshold because it might be impossible to produce the query results within the specified threshold. For example, the user cannot specify 1 millisecond as the threshold for a query involving tens of relations and producing millions of tuples. To help the user to select the reasonable time threshold, we provide the minimal

range which covers all values in the distribution table (e.g., Table 4.1) to the user, and let the user select a value in this range which best describes his requirements.

### 4.3.2 Workload Execution Time Threshold

In this section, we consider a different but also very fundamental target regarding a workload $\mathcal{Q}$, i.e., a batch of queries: finish as many queries as possible within $\tau$. We denote such an objective by *workload execution time threshold*. Note that since the distributions describes the execution times in a probabilistic way, it is impossible to exactly know the number of queries that can be finished within a certain duration. However, with carefully designed plan selection strategy, we are able to increase the expected number of queries to finish within $\tau$. In the following, we formalize the workload execution time threshold objective by modelling it as an integer programming problem, and propose a simple yet intuitive plan selection heuristic to solve the optimization problem.

We consider a particular query $Q_i$. Assume there are $n_i$ different plans for $Q_i$, and let $v_i^j \sim D_i^j$ be the variable denoting the execution time of the $j$-th plan ($j \in [1, n_i]$), where $D_i^j$ is the corresponding execution time distribution. We use vector $V_i = [v_i^1, v_i^2, \cdots, v_i^{n_i}]$ to organize all the execution time variables of $Q_i$'s plans. Let $C_i$ be a binary vector, with $C_i[j]$ being 1 if the $j$-th plan is selected as the execution plan for $Q_i$ and all other bits being 0. Increasing the number of queries in the workload that can be finished within the given threshold $\tau$ is essentially solving the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \int \cdots \int_{t_1,\cdots,t_m \in (0,\tau]} |S_\tau(t_1, \cdots, t_m)| \times \\ & \prod_{i \in S_\tau(t_1,\cdots,t_m)} P(C_i \cdot V_i^\intercal \leq t_i) dt_1 \cdots dt_m \\ \text{subject to} \quad & \forall i \in [1, m],\ C_i^\intercal C_i = 1 \end{aligned} \tag{4.2}$$

where $S_\tau(t_1, \cdots, t_m) \subseteq \{1, \cdots, m\}$ and $\sum_{i \in S_\tau(t_1,\cdots,t_m)} t_i \leq \tau$, and $\forall S$ s.t. $S \subseteq \{1, \cdots, m\} \wedge \sum_{i \in S} t_i \leq \tau$, $|S_\tau(1, \cdots, m)| \geq |S|$. In other words, $S_\tau(t_1, \cdots, t_m)$ is the maximal set of subscripts in range $[1, m]$ such that the sum of values in $\{t_1, \cdots, t_m\}$ indexed by these subscripts does not exceed $\tau$.

The constraint of Equation (4.3) guarantees that one and only one plan is selected for each query. Each selected plan $P_i$ is assigned a "local" time threshold $t_i$ denoting the time it should finish, and in order to finish more plans within $\tau$, we first execute these plans associated with smaller $t_i$. For a particular configuration of local time thresholds, $t_1, \cdots, t_m$, we only consider the plans indexed by $S_\tau(t_1, \cdots, t_m)$ as these queries are expected to be the fastest and adding any of the remaining plans may break the overall time threshold constraint. The probability that plans indexed by $S_\tau(t_1, \cdots, t_m)$ can finish within their corresponding local time threshold can be computed as $\prod_{i \in S_\tau(t_1, \cdots, t_m)} P(C_i \cdot V_i^\intercal \leq t_i)$, and the expected number of plans that can be finished within $\tau$ is thus $|S_\tau(t_1, \cdots, t_m)| \times \prod_{i \in S_\tau(t_1, \cdots, t_m)} P(C_i \cdot V_i^\intercal \leq t_i)$. The target is to select the best plan for each query (choosing $C_i$) such that the expected number of plans that can be finished within $\tau$ across all possible configurations of local time thresholds can be maximized.

Optimizing Equation (4.3) is an integer programming problem (choosing a plan id for each query) which is known to be NP-complete. Since it is too expensive to exhaustively enumerate all plan combinations, next we propose an efficient and effective heuristic for plan selection.

In order to maximize the expected number of plans finished within $\tau$, there are two questions we need to answer: (1) in what order do we process the queries, and (2) which plan to choose for each query. We propose Algorithm 4 that answers the two questions simultaneously.

The intuition behind Algorithm 4 can be described as follows. Suppose that we have constructed estimators under probability levels $q_1, q_2, \cdots, q_k$ (sorted in descending order). We first use the model corresponding to probability $q_1$ to estimate the execution time of each candidate plan for each query in the workload and choose the plan with the minimal estimated execution time. We sorted the selected plans in ascending order their estimated execution times (so that plans with low estimated execution time can be executed first). We count the number of plans that can finish within $\tau$, which is assumed to be $k$, and since with the current probability level, only $q$ (in percentage) plans could finish within their estimated execution times, we know the expected number of plans that can finish within $\tau$ can be approximated as $k * q$. We traverse over all probability levels and find the collection of plans

**Algorithm 3** Plan Selection: Workload Execution Time Threshold

**Input:** Probabilities in descending order $q_1, \cdots, q_k$, models corresponding to each probability level, workload $\mathcal{Q}$, $\tau$
**Output:** A sequence of plans
**Initialization:** FSP=$\emptyset$, $T = 0$
**for** $q = q_1, \cdots, q_k$ **do**
    Let $\mathcal{L}$ be the model corresponding to probability level $q$;
    Let $S = \emptyset$;
    **for** $Q \in \mathcal{Q}$ **do**
        Use $\mathcal{L}$ to estimate the execution time of each candidate plan for $Q$;
        Let $P$ be the plan with the minimal estimated execution time;
        Add $P$ to $S$;
    **end for**
    Let SP be plans in $S$ in ascending order of execution times;
    Find $k$ such that $\sum_{v \in \text{SP}_{[:k]}} v \leq \tau$ and $\sum_{v \in \text{SP}_{[:k+1]}} v > \tau$
    **if** $k * q > T$ **then**
        $T = k * q$;
        FSP=SP
    **end if**
**end for**
**return** FSP;

which lead to the maximal expected number of plans that could finish within $\tau$.

### 4.3.3 Workload Percentile Objective

In this section, we consider another practical and interesting objective regarding a workload. Again assume that there are $m$ queries in the workload, labeled $Q_1, \cdots, Q_m$. The objective is that for the given workload, at least $M$ ($M \in [1, m]$) queries should finish within duration $\tau$, which is denoted by *Workload Percentile Objective*. Similar to the problem introduced in Section 4.3.2, it is impossible to know whether a provided percentile objective can be satisfied, and thus the target here is to increase the probability that the percentile objective can be satisfied. In the following, we reuse the notations defined in Section 4.3.2 (including $C_i$ and $V_i$) and similarly model the workload percentile objective as an integer programming problem. More specifically, increasing the probability that the at least $M$ queries can be

finished within the given threshold $\tau$ is essentially solving the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \int \cdots \int_{\substack{t_1,\cdots,t_m \geq 0 \\ t_1+\cdots+t_m=\tau}} \prod_{i=1}^{m} P(C_i \cdot V_i^{\intercal} \leq t_i) dt_1 \cdots dt_m \\ \text{subject to} \quad & \forall i \in [1,m],\ C_i^{\intercal} C_i \in \{0,1\}, \text{and} \sum_{i=1}^{m} C_i^{\intercal} C_i = M \end{aligned} \tag{4.3}$$

The constraint of Equation (4.3) guarantees that at most one plan is selected for each query ($C_i^{\intercal} C_i \in \{0,1\}$), and the total number of selected plans is $M$ ($\sum_{i=1}^{m} C_i^{\intercal} C_i = M$). For the other $m - M$ queries, we can adopt any metric for plan selection, and they are always executed after the $M$ plans selected by Equation (4.3) and thus has no influence on the probability that the percentile goal can be satisfied. The objective of Equation (4.3) can be described as follows: if a plan is selected for query $Q_i$ ($C_i^{\intercal} C_i = 1$), we assign it a "local" threshold $t_i$ and compute the probability that the selected plan finishes within $t_i$ ($P(C_i \cdot V_i^{\intercal} \leq t_i)$). We conduct the computation for all queries and ensure all local thresholds sum up to $\tau$. Note that if $C_i^{\intercal} C_i = 0$, then $P(C_i \cdot V_i^{\intercal} \leq t_i)$ is always 1. The integral over all possible combinations of local thresholds is the probability to satisfy the percentile objective of the workload.

Since solving Equation (4.3) is equivalent to answer an integer programming problem (choosing a plan id for each query), we design a heuristic for plan selection. More specifically, we propose algorithm 4 that decides in which order to process the queries and which execution plan to use for each query.

The intuition behind Algorithm 4 can be described as follows. Assume that we have estimators under probability levels $q_1, q_2, \cdots, q_k$ (sorted in descending order). We first use the model corresponding to probability $q_1$ to estimate the execution time of each candidate plan of queries in the workload and choose the plan with the minimal estimated execution time for each query. We sorted the selected plans in ascending order their execution times (so that plans with low execution times can be executed first, to maximize the probability to satisfy the objective). If the sum execution time of the first $M$ plans in the sorted sequence is smaller than $\tau$, we execute these plans in the same order. However, if the sum execution time is larger than $\tau$, meaning that the percentile objective cannot be satisfied at

**Algorithm 4** Plan Selection: Percentile Objective

**Input:** Probabilities in descending order $q_1, \cdots, q_k$, models corresponding to each probability level, workload $\mathcal{Q}$, $M$, $\tau$
**Output:** A sequence of plans

```
for q = q_1, ..., q_k do
    Let L be the model corresponding to probability level q;
    Let S = ∅;
    for Q ∈ Q do
        Use L to estimate the execution time of each candidate plan for Q;
        Let P be the plan with the minimal estimated execution time;
        Add P to S;
    end for
    Sort plans in S in ascending order of execution times;
    Let C be sum execution time of the first M plans;
    if C ≤ τ then
        return plans in the same order;
    end if
end for
```

the current probability level, we repeat the above process with the model corresponding to the next probability level.

### 4.3.4 Minimizing the Overall Execution Time of Query Batch

Minimizing the overall execution time of a query batch is of great interest in most scenarios. With the execution time distribution of all plans, we have a different and more accurate approach to select the fastest plan for each query: computing the expected execution time of each plan based on their execution time distributions to identify the one with the shortest latency. Using the expected execution time reduces the bias introduced by a single cost estimator, and helps to identify plans with stable performance, which leads to lower query answering latency, as will be shown in Section 4.5.6.

## 4.4 Integration to DBMS

In this section we introduce the integration of the proposed plan selection strategy into DBMS. More specifically, we present the process of building the distribution constructor ($DC$) offline, and utilizing $DC$ and the strategies designed in Section 4.3 for plan selection,

as well as the steps to force the DBMS executes the selected plan. We also perform a concrete case study using PostgreSQL to ease the understanding of the integration procedure. As will be shown later in this section, integrating our method into DBMS is very lightweight, requiring no modification of the DBMS and incurring only minor overhead to the query processing procedure.

### 4.4.1 Building Distribution Constructor

In Algorithm 1 we summarized the process of constructing the distribution constructor $DC$, which serves as the basis of online plan selection. In this section, we present more details regarding the inputs of Algorithm 1, especially the generation of samples $\{(P_i, t_i)\}_{i=1}^n$, and how to build $DC$ from these samples.

Let $\mathcal{Q}$ be a query workload, which can be the queries provided together with the benchmark dataset, or the queries in the DBMS log. In order to provide a rich set of $\{(P_i, t_i)\}_{i=1}^n$ samples, for each query $Q \in \mathcal{Q}$, we feed the DBMS multiple sets of query hints $\mathcal{H}_1, \mathcal{H}_2, \cdots, \mathcal{H}_n$, producing $n$ candidate plans to execute $Q$, $P_1, P_2, \cdots, P_n$. Query hint is a feature generally supported by existing DBMSs such as PostgreSQL, MySQL and SQL Server, and manipulates the plan generation process by enabling/disabling certain operators such as Index Scan and Hash Join. Each hint set consists of one or more such hint, and providing a hint set to the DBMS so that the DBMS generates the execution plan only using enabled operators. Therefore, different combinations of hints result in different execution plans of $Q$. We adopt the same sets of hints designed in [116]. The plan executed by the DBMS can be retrieved using command "EXPLAIN $Q$", which prints the execution plan of $Q$ in a desired format (e.g., JSON, XML), and the execution time is obtained by actually running the plan, forming a $(P_i, t_i)$ pair. Applying the hint sets to (all or partial) queries in $\mathcal{Q}$ thus results in a collection of samples $\{(P_i, t_i)\}_{i=1}^n$, which serve as the input of Algorithm 1 to build the distribution constructor.

As is clear from Section 4.2.3, $DC$ is built on top of the execution time estimator $\hat{\mu}$, which consists of a cost estimator and optionally a plan encoder and a cost-execution time mapper. We show the choice of each component below. For a particular DBMS, the cost estimator is fixed. For example, it can be either the default cost estimator of the DBMS, or

an external learned cost estimator specified by the DBMS user. We can adopt techniques in the literature designed for plan encoder [153, 112] and cost-execution time mapper [180] to reduce extra labor, and assembling the three components produces $\hat{\mu}$, as shown in Figure 4.2.

The process of calibrating $\mu$ using $\{(P_i, t_i)\}_{i=1}^{n}$ to construct $DC$ (line 6 of Algorithm 1) depends on the conformal technique, and some techniques simply calculate and leverage the absolute differences between $\hat{\mu}(P_i)$ and $t_i$ while others involve ML models and training steps. As will be presented in Section 4.5.1, we conduct experiments with multiple conformal techniques to identify the one that best suits the task studied herein.

### 4.4.2 Query Processing

Given the distribution constructor, we can illustrate the query processing in Figure 4.5. The work flow of Figure 4.5 can be described as follows:

1. **Candidate plan generation.** Let $Q$ be a new query to be executed, we feed the DBMS the same sets of query hints as in the offline preparation step (Section 4.4.1) together with $Q$, producing $n$ candidate plans to execute $Q$, $P_1, P_2, \cdots, P_n$. Note that generating $n$ plans using hints invoke the DBMS planning procedure $n$ times, the overhead of which, however, is minor compared with the improvement in query performance resulted from our design, as will be shown in Section 4.5. We then extract the plans using the "EXPLAIN" command again.

2. **Distribution construction.** Candidate plans of $Q$ are sequentially fed into the distribution constructor, and the execution time distribution of these plans, $D_1, D_2, \cdots, D_n$ are produced, as described in Algorithm 2.

3. **Plan selection.** The plan selector consists of the objectives designed in Section 4.3 and the DBMS user can determine which objective to use based on his own requirement for the task. The plan selector takes $D_1, D_2, \cdots, D_n$ as input, and identifies the plan that best satisfies the selected objective, say $P^*$. Since $P^*$ is generated by the DBMS when hint set $H^*$ is used, we feed $H^*$ to the DBMS together with query $Q$, and the

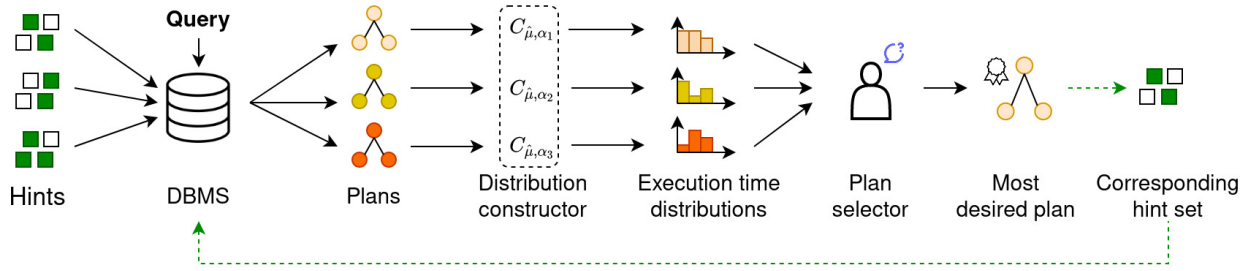



Figure 4.5: Integration to DBMS

DBMS would generate $P^*$ again as the execution plan for $Q$, in line with the user's preference (green dashed line).

As is clear from above description, applying our design into an existing DBMS incurs no change to the DBMS and only minor change to the query answering process by adding a lightweight query pre-processing stage.

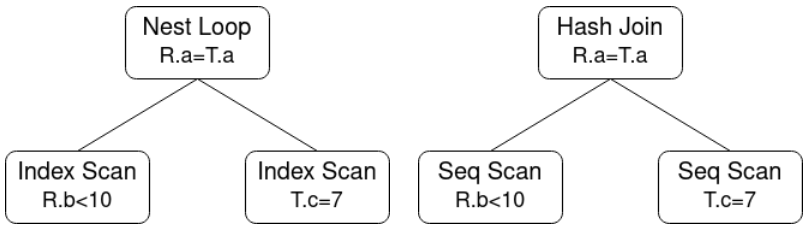
### 4.4.3 Case Study with PostgreSQL

We conduct a case study regarding the integration of our proposal into DBMS using PostgreSQL as the example. PostgreSQL supports setting the following switches to on and off, which controls the operators allowed in generating query plans: {*enable_seqscan, enable_indexscan, enable_indexonlyscan, enable_bitmapscan, enable_nestloop, enable_hashjoin, enable_mergejoin*}. Note that all switches are on by default.

We consider query $Q$: "*select * from R, T where R.a=T.a and R.b¡10 and T.c=7*",and two hint sets: $\mathcal{H}_1$ ={*set enable_hashjoin=off, set enable_indexscan=off*}, $\mathcal{H}_2$ ={*set enable_nestloop=off, set enable_seqscan=off*}. After each hint set is applied to PostgreSQL, we can use the "EXPLAIN" command to print the plan thus generated without executing it. Assume applying $\mathcal{H}_1$ and $\mathcal{H}_2$ lead to the plans shown in Figure 4.6(a) and Figure 4.6(b), denoted by $P_1$ and $P_2$ respectively.

We then adopt Algorithm 2 to construct the execution time distribution for each plan, which is given in Table 4.3.

Assume that the user's objective is to get the result of $Q$ within 8s, then from Table 4.3 we know that $P_2$ is preferred as $P_2$ has 90% probability to finish within 8s, while $P_1$ has




(a) Plan $P_1$ (b) Plan $P_2$

Figure 4.6: Two Plans

Table 4.3: Execution Time Distribution of $P_1$ and $P_2$

| $\alpha$ | Probability level | Interval of $P_1$ | Interval of $P_2$ |
|---|---|---|---|
| 0.1 | 90% | [1s,9s] | [1s,8s] |
| 0.2 | 80% | [1s,8s] | [2s,7s] |
| 0.3 | 70% | [2s,5s] | [2s,6s] |
| 0.4 | 60% | [3s,4s] | [3s,5s] |

80%. Therefore, hint set $\mathcal{H}_2$ is passed to PostgreSQL together with $Q$, and the DBMS will process $Q$ using plan $P_2$. In another case, assume that the user's time limit is 5s, then $P_1$ is preferred, as $P_1$ has 70% probability to finish within 5s while $P_2$ has 60%, and hint set $\mathcal{H}_1$ should be passed into the DBMS.

## 4.5 Experiments

In this section, we study the effectiveness of the distribution-based plan selection strategies defined in Section 4.3 in satisfying the corresponding objectives. We conduct extensive experiments on four benchmarks, with open source DBMS and commercial DBMS, and using both conventional cost estimator and learned cost estimator.

### 4.5.1 Settings

**Datasets.** We adopt four open-source benchmarks used for the task of query optimization with various size and complexity, namely JOB-light [85], CEB [126], and Stack [116]. JOB-light and CEB are based on the IMDb dataset, Stack consists of questions and answers from Stack Exchange websites. The statistics of the datasets are summarized in Table 4.4.

Table 4.4: Dataset Characteristics

| | JOB-light | CEB | Stack |
|---|---|---|---|
| Size | 10GB | 10GB | 100GB |
| Num. Queries | 10,000 | 10,000 | 6,000 |
| Query Type | Numeric | Numeric+String | Numeric+String |
| Max num. relations in a query | 3 | 16 | 8 |
| Avg num. relations in a query | 2.2 | 9.6 | 5.6 |

**Cost estimator.** We implement the design with PostgreSQL and a commercial database (denoted by DBX) and build the execution time distribution based on their cost estimation components. Default parameter settings are used for both databases. In order to testify the effectiveness of the proposed methods when integrating into other types of cost estimators, we also have conducted experiments using learned cost estimators. In this work we report the results using the estimator in [116], a lightweight yet well-performed learned model, and observations with other models [153] are similar. Details regarding calibrating learned cost estimator can be found in Section 4.5.8.

**Conformal technique.** We have conducted experiments with multiple conformal techniques [**RomanoPC19TagasovskaL19**, 87] and in the experiment we report results obtained using the technique in [87] as it yields the best performance for the task studied in this work. The conformal technique in [87] consists of a learned module, and we use the network in [116] to build this module for its simplicity. The investigation of other conformal techniques and models to fit the relation from a plan to the estimation error would be interesting future works.

**Implementation and environment.** The estimation calibrator are implemented in Python (version 3.8.10), and we use PostgreSQL (version 12.9) or DBX as the engine to execute the queries, depending on which DBMS the calibrator is associated with. For experiments on the learned estimator, we use PostgreSQL as the back-end DBMS (note that the choice of DBMS does not influence the conclusions thus derived). Experiments are con-

ducted on a Ubuntu 20.04 instance with Core i5 CPU, 24GB RAM, 256GB SSD, and 2TB HDD. Experiments with TPC-H are conducted on HDD as the dataset is too large to fit in SSD, and other experiments are conducted on SSD.

**Evaluation.** We compare the performance of the cost estimator before and after calibration in achieving each of the objectives designed in Section 4.3. For each workload, we randomly select 3000 queries as the evaluation set, and the remaining queries are used to initialize the estimator calibrator (as will be introduced in Section 4.5.2). We repeat each experiment 5 times to compute the average execution time. Please note that the calibrated estimator incurs extra DBMS planning time and model inference time, which are included in the overall query answering time of the method.

### 4.5.2 Correctness of the Execution Time Distribution

The foundation of the framework is the correctness of the execution time distribution, more specifically, the predefined probability level $1-\alpha$ in the conformal technique being close to the actual coverage rate (ACR) at the same probability level, which is computed as follows:

$$\text{ACR} = \frac{\sum_{P_i \in \mathcal{P}} [t_i \in [l_i^\alpha, u_i^\alpha]]}{|\mathcal{P}|} \tag{4.4}$$

where $\mathcal{P}$ denotes the set of plans executed in the evaluation phase, $P_i$ is a query in $\mathcal{P}$ and $t_i$ is the execution time of $P_i$, and $[*]$ is the indication function which equals to 1 if the statement $*$ is true and 0 otherwise.

Only with accurate execution time distribution (ACR$\approx 1-\alpha$) will the plan selection based on the execution time distribution be valid. As introduced in Section 4.2.1, the application of the estimator calibrator involves an initialization stage when execution plans together with their costs and execution times (referred to as samples) are passed to the technique. Theoretically, using more initialization samples would provide more comprehensive information regarding the data and the ACR becomes closer to $1-\alpha$. In this section, we study the correctness of the execution time distributions in terms of the number of samples used for initialization, and report results of $\alpha = 0.1$ in Figure 4.7. The observations with other values of $\alpha$ are similar and are thus omitted for brevity.

The horizontal axis of Figure 4.7 denotes the number of initialization samples, while the vertical axis denotes the absolute difference between the predefined probability level and the ACR. As can be observed from Figure 4.7, as the number of initialization samples increases, the absolute difference between ACR and $1-\alpha$ decreases quickly at first and becomes stable at 1,000 to 2,000 samples for all benchmarks and databases. Note that it is theoretically impossible to have an absolute difference equaling to zero due to the data variance and limitation of the cost estimator and the conformal technique. However, as will be shown in the following sections, the plans thus selected yield good performance, despite of the minor difference between the actual coverage rate and expected coverage rate.

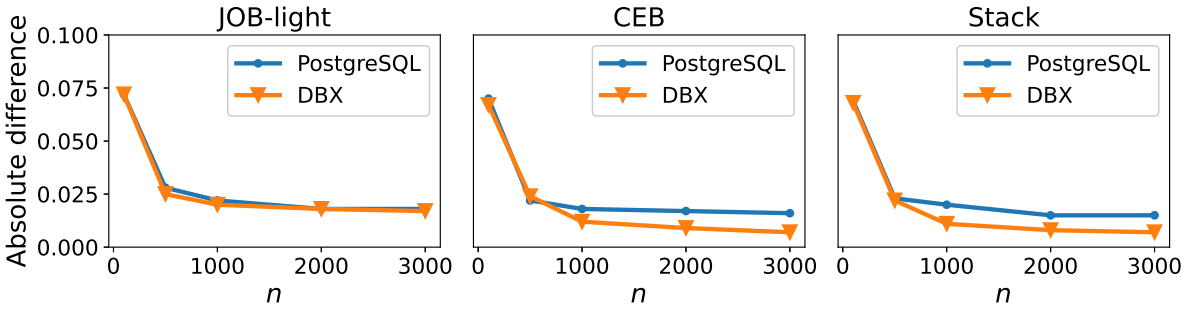


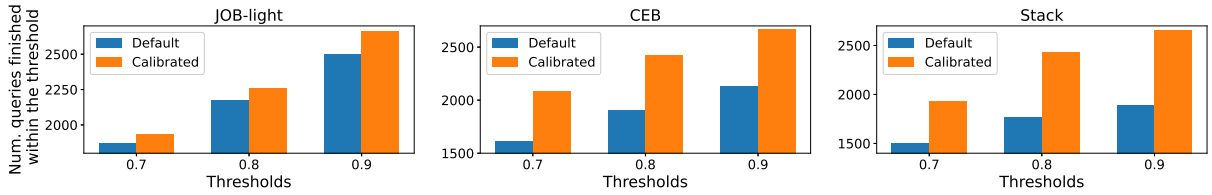

Figure 4.7: Coverage Rate

### 4.5.3 Per-query Time Threshold

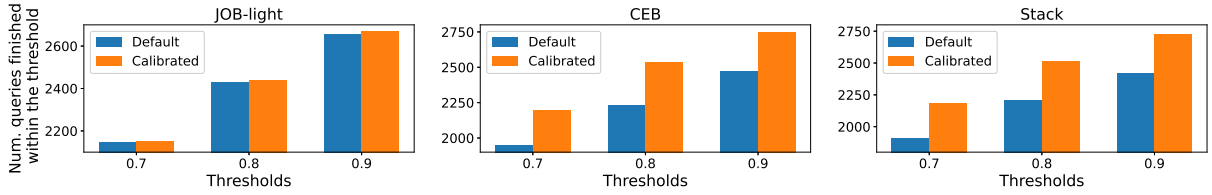
In this section, we examine the performance of the plan selection strategy designed in Section 4.3.1 in finishing more queries within their assigned thresholds. Particularly, for each query in the evaluation set, we set a time threshold, and report the number of queries that can finish within the threshold. As described in Section 4.3.1, we construct a reasonable range (denoted by $R$) to select the threshold $\tau$ based on the execution time distribution. We vary the values of $\tau$ to validate the effectiveness of the method in dealing with various thresholds. The results for $\tau$ being the $\{0.7, 0.8, 0.9\}$ quantile of $R$ are reported in Figure 4.8. The observations with other values of $\tau$ are similar and are thus omitted for brevity.

As can be observed from Figure 4.8, with the execution time distribution and the corresponding plan selection strategy, more plans that can be finished within the specified threshold, compared with the default optimizer of PostgreSQL and DBX and across differ-

ent benchmarks. More specifically, we have the following observations: (1) The proposed method outperforms PostgreSQL and DBX, and lead to up to 25% more plans in the evaluation workload that can be finished within the corresponding time threshold. The reason is that, with the execution time distribution we are able to compute the probability each plan finishes within a certain time, and by selecting the plan with the highest probability to finish within the assigned limit rather than the plan with the lowest estimated cost (which might be inaccurate), we have higher chance to satisfy the threshold constraint; (2) The superiority of our method is more significant on CEB and Stack than on JOB-light. The reason is that, compared with JOB-light, the queries of CEB and Stack are more complex (involving both string and numerical data types and consisting of more join operations), as shown in Table 4.4. As a result, the estimations of the default optimizer regarding queries and plans on CEB and Stack are less accurate, and calibrating the estimator to produce execution time distribution results in higher improvement in query performance, which also proves the effectiveness of the proposed method in dealing with complex workloads.




(a) Postgres




(b) DBX

Figure 4.8: Per-query Time Threshold

### 4.5.4 Workload Time Threshold

In this section we compare the default estimator and calibrated estimator in achieving the workload time threshold objective under various thresholds. More specifically, given the evaluation set of each dataset, we use different values of $\tau$ as the threshold, and count

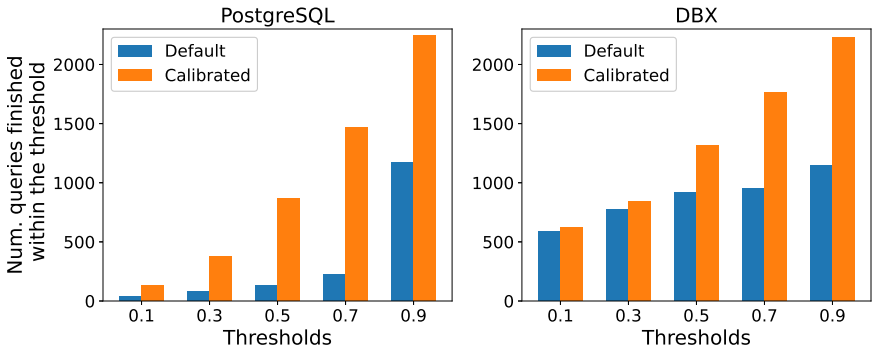



Figure 4.9: Workload Time Threshold

the number of queries in the set that can be finished in total within the threshold. Let $T$ be the minimal duration to finish all queries in the workload, we select $\tau$ in range $[0.1T, 0.3T, 0.5T, 0.7T, 0.9T]$. We report observations on CEB in Figure 4.9. The results on other datasets are similar and omitted for brevity.

It is clear from Figure 4.9 that using the execution time distribution and the algorithm designed in Section 4.3.2 for plan selection outperforms the default optimizer across different settings and finishes up to 5 times more queries within the specified time threshold. The reason is that, by selecting execution plans for each query and choosing the execution order based on their execution time distributions as described in Algorithm 3, our method strives to maximize the expected number of queries finished within the given time threshold. With the default optimizer, although plans with low estimated costs are executed first, such estimations may be inaccurate and plans with low estimated cost but high actual execution time would slow down the entire execution process, leading to lower number of queries finished in the threshold. Note that for the case when the value of $\tau$ is very small (e.g., 0.1) or very large (0.9), corresponding to the case when the threshold is too small to finish even a intermediate number of queries and the case when the threshold is close to the overall execution time of the entire workload, the advantage of our method becomes less significant. However, for a wide range of thresholds, calibrating the estimator results in much better query performance in terms of finishing more queries within the specified threshold.

### 4.5.5 Workload Percentile Objective

In this section we testify the performance of the methods in dealing with percentile objectives. As described in Section 4.3.3, percentile objective is that for a given workload $\mathcal{Q}$, at least $M$ queries must finish within duration $\tau$. To examine the effectiveness of the method across various settings, for the same evaluation set, we vary the values of $M$ and $\tau$ and count the number of times the objectives can be satisfied. For practical consideration, we vary $M$ in range [0,1] denoting the percentage of queries to finish, and for a particular $M$, let $T$ be the minimal duration to finish $M$ queries, and we randomly sample the value of $\tau$ in three ranges: $[1T, 1.2T]$, $[1.2T, 1.4T]$, $[1.4T, 1.6T]$, corresponding to the case of hard objective, intermediate objective, and easy objective. Note that further reducing or increasing the value of $\tau$ are not necessary to compare the methods, as will be discussed below. We repeat the experiments with 50 different workloads (randomly sampled with ratio 0.5 from the evaluation workload of JOB-light) and report the total number of workloads which satisfy the corresponding percentile objective.

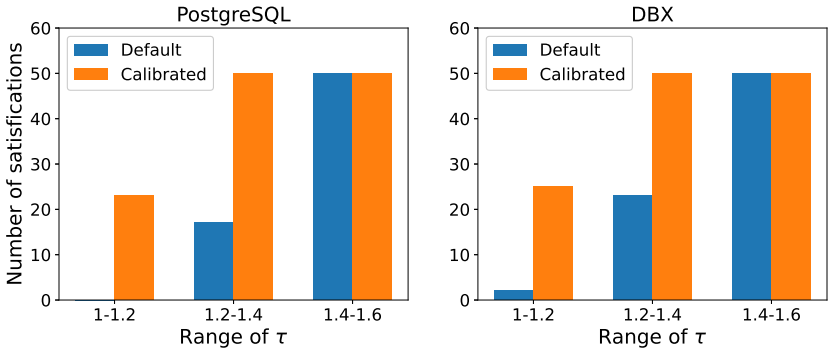



Figure 4.10: Percentile Time Threshold

In Figure 4.10 we report the results obtained with $M = 0.5$, and observations with other values of $M$ are similar. As can be observed from the figure, increasing the value of $\tau$ leads to higher number of workloads satisfying the objective, and our method outperforms default estimator across various settings. The reason is that, by using the heuristic introduced in Algorithm 4 to maximize the probability that at least $M$ queries finish within duration $\tau$, our method increases the chance the percentile objective to be satisfied, while with the

default estimator, plans with low estimated cost (and are thus selected to be among the first $M$ queries to execute) but high actual execution time may slow down the execution process, reducing the probability to satisfy the percentile objective. Since sampling $\tau$ in range $[1T, 1.2T]$ gives hard objective (almost no workloads satisfying the objective with the default estimator) and sampling $\tau$ in range $[1.4T, 1.6T]$ gives easy objective (all workloads satisfying the objective with the default estimator), further reducing or increasing the value of $\tau$ is not necessary to compare the methods.

### 4.5.6 Minimizing Overall Execution Time

Minimizing the overall execution time of a query batch is of great interest in most scenarios. Although the framework proposed herein is not directly optimized for the overall execution time, constructing the execution time distribution provides a new perspective of minimizing the overall execution time: compute the expected execution time of each plan based on the execution time distribution, and select the one with the minimal expected execution time as the execution plan. The comparison of the expected execution time-based strategy and the default estimator is given in Figure 4.11. Note that the extra query planning time and model inference time incurred by the proposed method are included in the results.

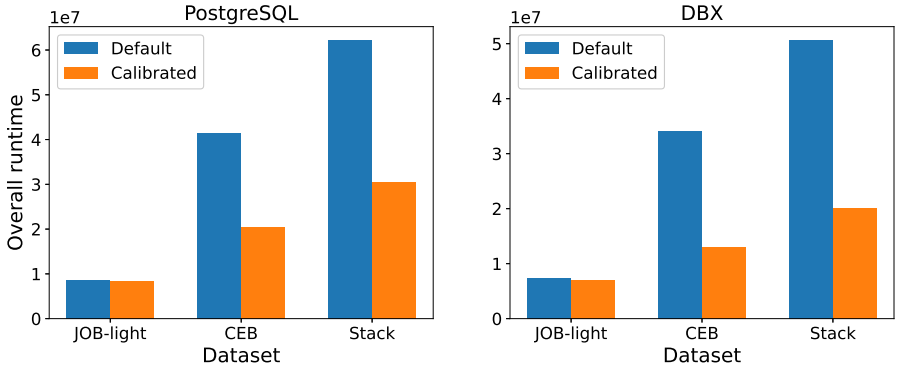



Figure 4.11: Workload Execution Times

As shown in Figure 4.11, using the execution time distribution for plan selection to minimize the overall execution time reduces the query answering time by up to 60%. The reason is that, using the execution time distribution to compute the expected cost of a plan

greatly reduces the bias of a single estimated value, and thus the overall execution time can be significantly reduced compared with the default estimator, especially for complex workloads such as CEB and Stack when estimation bias is more likely to occur due to data variance.

### 4.5.7 Influence of the Distribution Granularity

As introduced in Section 4.2.2, we construct the execution time distribution by varying the value of $\alpha$ in the conformal technique, and the number of $\alpha$s used during the process, i.e., the granularity of the constructed distribution, has an impact on the query performance. Intuitively, using more $\alpha$s would produce more fine-grained and accurate distributions, at the cost of longer query pre-processing time to construct the distribution. In this section, we study the influence of the distribution granularity on the query performance. When using $k$ $\alpha$s, we choose the values with equal width in range [0,1]. For example, when $k = 3$ we use $\alpha \in \{0.25, 0.5, 0.75\}$. We report the results using per-query time threshold on JOB-light in Figure 4.12, and similar conclusion can be obtained with other objectives and settings.

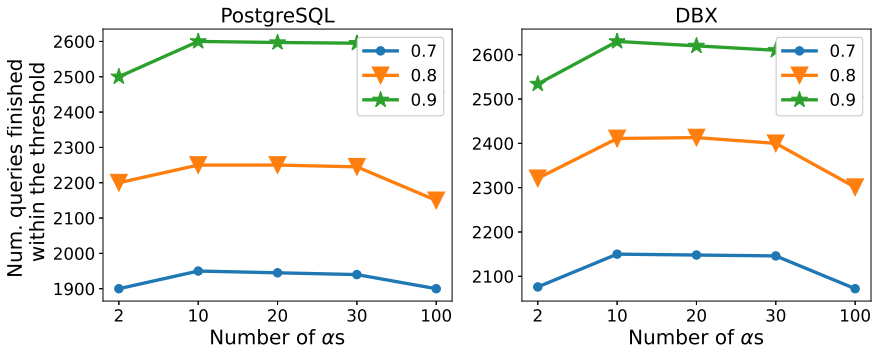



Figure 4.12: Influence of Distribution Granularity

As can be observed from Figure 4.12, using 10 $\alpha$s results in the maximal number of queries finished within the corresponding threshold for per-query threshold objectives. An overly coarse-grained distribution cannot accurately describe the execution time performance, and the plan thus selected would be sub-optimal. While increasing the number of $\alpha$s leads to more accurate distributions, constructing the distribution at query time results in higher overhead,

slowing down the query answering. Therefore, in practice we recommend a distribution granularity with 10 $\alpha$s.

### 4.5.8 Studying the Learned Estimator

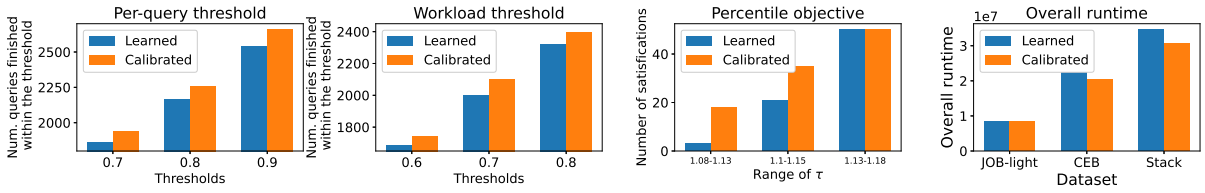



Figure 4.13: Calibrating Learned Estimator

In this section we study the effectiveness of the proposed method in calibrating learned estimators, which have attracted much research interests and shown superior performance across various benchmarks. Since the estimation calibrator and plan selection strategy designed in our work treat the cost estimator as a black-box and only depend on the estimation results, the proposed methods can be applied to learned estimator without any modification. More specifically, we assume that the learned estimator is already trained on the target workload, and each time the cost of a plan is needed by the calibrator, instead of using the output of the default estimator, we use the output of this learned estimator. Other parts in query answering remain the same as depicted in Section 4.4. We use PostgreSQL as the back-end query processing engine, and the choice does not influence the observations made in this section. We report results obtained using the learned estimator designed in [116] and with CEB, and experiments with other model structures [153] and workloads yield similar observations. The results are presented in Figure 4.13.

From Figure 4.13 we have the following key observations:

1. For per-query threshold objective and workload threshold objective, calibrating the learned estimator and utilizing the corresponding plan selection strategy increases the number of queries that can finish within their assigned threshold in all settings;

2. For percentile objective, the execution time distribution-based plan selection strategy increases the number of workloads that satisfy the objective by up to 5 times, for threshold $\tau$ in certain ranges;

3. Using expected execution time to identify the fastest plan reduce the overall execution time of the workload by up to 15% compared with merely using the estimation of the learned cost estimator.

Comparing Figure 4.13 to results obtained with the default DBMS estimator in previous sections, we can observe that calibrating the default estimators results in higher improvement in query performance. The reason is that, as shown in previous studies [172], ML models are adaptive to the data and better captures the mapping relation from plans to execution times, and thus are usually more accurate for cost estimation than conventional statistics-based cost estimators. As a result, for the same query, learned estimator has higher chance to identify a better plan than conventional estimator, and thus conventional estimator would benefit more from calibration. However, as can be observed from Figure 4.13, the improvement of calibrating the learned estimator in different objectives are all significant, proving the effectiveness of the proposed method in applying to various types of cost estimators.

## 4.6 Summary

In this work we have studied the problem of constructing the execution time distribution for query plans and leveraging the distribution for plan selection to fulfill the DBMS user's various objectives regarding query performance. We enhanced conformal prediction technique to produce execution time distributions for arbitrary plans, which may be integrated into any types of plan performance predictors incurring very minor overhead. We have conducted extensive experiments on multiple benchmarks, and the results validated the effectiveness of the proposed method in achieving various objectives regarding query performance.

Note that although in this work we mainly expanded our discussion around relational DBMS, the techniques thus developed can be naturally generalized to non-relational DBMS and other systems, and the cost measure is not limited to execution time. For example, we can construct the distributions of travelling time of candidate paths in a traffic scheduling system. The deployment of the technique in practical systems with various cost measures would be an interesting and important direction for future investigation.

In this work we adopt conformal prediction to construct distributions as it provides theoretical guarantees regarding the coverage, and researchers have developed various techniques for distribution construction. For example, the technique in [100] partitions the domain of the target variable into bins and estimates the conditional density function using multi-class classification methods; the method in [68] applies a logistic transformation on the output layer of the model to be calibrated, which produces an expression of the conditional density distribution, and the authors has studied the effectiveness of the method when used for both polynomial regression and deep learning models. The investigation of applying various methods for the task of execution time distribution construction would be an important direction for future investigation.

# 5 Data Acquisition for Improving Machine Learning Models

## 5.1 Introduction

### 5.1.1 Background and Motivation

Data traditionally has been an asset in deriving projections or making decisions. The prevalence of Machine Learning (ML) models across business functions necessitates access to ample and diverse data sources for training. As an answer to the vast demand for training data, numerous businesses offer data annotation services [4, 66, 5] providing annotated data in a myriad of business categories with varying degrees of specialization. It is evident that the need for specialized data to train ML models has created a corresponding market fulfilling the purpose.

At the same time, in recent years we have experienced the increasing prevalence of *online data markets* such as Dawex [25], WorldQuant [175], and Xignite [183], to name a few, which aim to make access to data a commodity, for modelling or learning purposes. In these markets the main idea is to facilitate interaction between *data providers* (e.g., individuals or organizations that possess data in diverse domains and wish to offer them to other interested parties) and *data consumers* who are interested to obtain data to accomplish certain tasks, such as training new ML models or increasing the accuracy of existing ones, or conducting statistical estimation. Since such platforms aim to adopt the characteristics of a market, data exchange carries an underlying cost (e.g., monetary value). The emergence of such markets can be viewed as an initial step to the enablement of efficient trading of data.

The design of the operating principles, market mechanisms and tradings strategies (to

name a few topics) of such markets constitute open research directions and involve multiple research communities. Recently, in the database community, Fernandez et al. [43] presented their vision for a research agenda on market design in data market platforms and discussed various important research directions of broader data management interest in making the data market vision a reality.

A problem related to the task studied herein is the online knapsack problem [59, 107], where a collection of items $e_i$ with values $v(e_i)$ and sizes $s(e_i)$ are given sequentially, and each time an item is given, the user needs to decide whether to keep it or not, with the objective of maximizing the total value of the selected items without violating the knapsack capacity constraint. The major difference between the online knapsack problem and the data acquisition problem is that, in online knapsack problem, the value of an item and the budget it consumes (i.e., the size of the item) are given so that users can make decisions accordingly, while in our setting, the value of the item is hidden and can only be revealed after the corresponding budget is consumed, which results in extra difficulty in designing the data acquisition strategy.

### 5.1.2 Problem Description

In this work, we consider a domain $\Gamma = \{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X}$ denotes the feature space and $\mathcal{Y}$ denotes the label space (e.g., possible class labels for classification tasks, possible values of the dependent variable for regression tasks). The purpose is to train a model for a target distribution $p$ over $\Gamma$ such that the model attains high accuracy on data drawn from the same distribution. We focus on supervised learning and assume that the data consumer (*consumer* for short) already has an ML model (e.g., CNN, SVM, a regression model) built utilizing some training data from $\Gamma$, and wishes to obtain data from a data provider (*provider*) offering $\mathcal{D}_{pool}$ drawn from the target distribution $p$. The aim of the consumer is to maximize the improvement in the accuracy of the model[6].

[6]We use the term "accuracy" in a broad sense here, which depending on the ML model can be measured in different ways (precision for classifiers, root mean squared error for regression models, etc.), and our discussion is independent of its exact choice.

To facilitate the interaction, the provider exposes meta-data of $\mathcal{D}_{pool}$, such as the range of values in each attribute, and the set of possible labels on the records. We assume a typical query interface supported by both parties, akin to the prevalent application programming interfaces (API) in existence for any online service [165]. The interface supports a predicate $P$ specifying the properties of the records requested, and an integer $I$ denoting the number of records to obtain (e.g., 10 images with `label = 'dog'`, or 100 records with $2018 \leq$ `year` $\leq 2020$). Such predicates impose multi-attribute conjunctive conditions in the more general case. After receiving the query request from the consumer, the provider randomly selects without replacement $I$ records satisfying $P$ from $\mathcal{D}_{pool}$ and returns these records to the consumer.

We assume that the consumer carries a *budget* $B$ on the total number of records that can be requested[7]. The number of records requested by the consumer each time a query is issued to the provider may vary, and is decided by the consumer, as long as the total number of records requested across all queries is within the budget $B$. For example, if the consumer can obtain 10 images from the provider, these can be obtained by inquiring for 10 images once or for 5 images twice.

Suppose that the accuracy of the underlying model is evaluated on a testing data set $\mathcal{D}_{test} \subset \Gamma$. The task of the consumer is to identify a series of queries $\langle (P_1, I_1), (P_2, I_2), \cdots, (P_z, I_z) \rangle$ to obtain $B$ records, where $\sum_{i=1}^{z} I_i = B$ with $P_i$ and $I_i$ being respectively the predicate and the number of requested records in the $i$-th query. Let $\mathcal{D}_{otd}$ be the records obtained from the provider using the identified queries ($|\mathcal{D}_{otd}| = B$), $\mathcal{D}_{init}$ be the data the model of the consumer is trained on initially, and $\mathcal{M}'$ be the model re-trained on all the data the consumer has after data acquisition (i.e., $\mathcal{D}_{otd} \cup \mathcal{D}_{init}$ ). The objective of the data acquisition process is to improve as much as possible the accuracy of $\mathcal{M}'$ on $\mathcal{D}_{test}$.

[7]Such a budget can be determined based on monetary costs per record offered by the provider or by the monetary cost of each query, etc. Any mechanism to assign a value to data (e.g., price or otherwise) is completely orthogonal to our approach.

### 5.1.3 Solution Overview

We develop data acquisition strategies to address this problem. In particular, we consider the trade-off between *exploration* and *exploitation* in data acquisition. During exploration, requested data records from the available budget are obtained to gain more knowledge regarding the distribution the provider's data, so that better predicates can be designed for subsequent queries. During exploitation, data records are obtained based on the current information the consumer possesses. With a limited budget of records to be requested, one must strike a balance between exploration and exploitation by allocating the requested records within the budget wisely such that the accuracy of the resulting model is maximized.

We propose two methods to determine how to allocate the existing budget of records across queries (the budget allocation problem) adopting different strategies. The first solution, which we refer to as *estimation-and-allocation* (EA), consists of two stages: during the Estimation Stage, the consumer issues a number of queries obtaining a number of random records for each of them to explore from $\mathcal{D}_{pool}$; we subsequently estimate (without re-training the model) the expected improvement in model accuracy utilizing the records for each query, which we refer to as *predicate utility*. During the Allocation Stage, the consumer allocates the remaining record budget according to the estimated utilities. We investigate methods to quantify the estimation error and propose an adaptive method to balance between reducing the estimation error and controlling how much of the record budget is devoted to obtaining the estimation; as a result, budget is reserved to be allocated more effectively for predicates with high utilities. For the Allocation Stage, we propose different allocation strategies and showcase their performance under various settings in Section 5.5.

The second solution, which we refer to as *sequential predicate selection* (SPS), is based on the observation that for a predicate $P$, the associated predicate utility decreases as we obtain more records for the predicate, due to information redundancy [54]. The core idea of SPS is to iteratively pose queries requesting a small number of records while balancing between (1) obtaining more records with predicates yielding higher expected utility, and (2) closely monitoring the utility decrease of each predicate as we obtain more records. We implement this design utilizing Thompson Sampling (TS) [139, 160], an action selection method, for its

simplicity and proven performance, but the design can be implemented with other action selection methods (such as the $\epsilon$-greedy algorithm [155]) as well.

As both EA and SPS rely on the expected predicate utility, we investigate how to best estimate it without re-training the underlying ML model. The utility of a predicate $P$ is essentially measured by the improvement in model accuracy resulting from the set of records selected by $P$, $\mathcal{R}_P$. We propose *novelty*, which describes how different the distribution of $\mathcal{R}_P$ is from the distribution of the records the consumer currently possesses satisfying $P$, as the indicator of the potential accuracy gain $\mathcal{R}_P$ brings to the model. Note that our subsequent discussion applies to other utility measures as well; we experimentally compare various measures in Section 5.5.11.

We evaluate the performance of EA and SPS on both traditional ML tasks and Deep Learning tasks, including spatial regression, radar data classification, and image classification, using classical ML models as well as state-of-the-art deep models. As will be shown in Section 5.5, the proposed methods demonstrate solid performance across a variety of settings, outperforming alternative approaches that require frequent model re-training. We also thoroughly study the effects of various parameters on EA and SPS and provide suggestions on their settings in real-world scenarios.

**Contributions.** Our main contributions can be summarized as follows.

- We formally define and study the problem of data acquisition for improving the performance of ML models given a budget. We consider this problem in the context of a data consumer and a data provider in a data market, and it can serve as a building block for a variety of data markets.

- We propose an estimation-and-allocation solution, EA, which first estimates the utility of each predicate with a portion of the budget, and then allocates the budget accordingly to improve the accuracy of the model.

- We devise a sequential predicate selection solution, SPS, which adaptively conducts exploration and exploitation, by iteratively requesting a small number of records in each query, aiming to improve the predicate utility estimates and utilize such estimates at the same time.

- We design methods to estimate the expected utility of a given predicate, allowing EA and SPS to proceed without necessitating the re-training of the underlying model.
- We experimentally study the proposed solutions across a variety of settings, including but not limited to, different tasks, ML models, datasets, distributions, budget limitations, utility measures. We showcase that each solution has certain benefits and can be suitably adopted when applied to real-world settings.

## 5.2 Preliminaries

In this section, we formally define the terminology utilized and the problems we focus on in this work.

**Data Domain and Learning Task.** We consider a supervised learning task defined on a data domain $\Gamma = \{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X}$ denotes the *feature* space and $\mathcal{Y}$ denotes the *label* space (e.g., all possible class labels for classification tasks, all possible values of the dependent variable for regression tasks). Suppose there is a conditional distribution $p(y|x)$ defined over $\Gamma$, where $x \in \mathcal{X}$, $y \in \mathcal{Y}$. The learning task is to train a model $\mathcal{M}$ on a dataset that represents a distribution $g$ that is as close as possible to the target distribution $p$. The accuracy of $\mathcal{M}$ is evaluated on a testing dataset $\mathcal{D}_{test} \subset \Gamma$. In accordance to any well-formed learning task, we assume that both the training data and testing data come from the same distribution $p$. The model $\mathcal{M}$ is evaluated using a function $F$ based on $\mathcal{D}_{test}$, denoted as $F(\mathcal{M}; \mathcal{D}_{test})$.

**Provider, Consumer, and Budget.** The provider maintains a collection of data records $\mathcal{D}_{pool} \subset \Gamma$ drawn from the target distribution $p$, which are provided in the data market and are initially entirely invisible to the consumer. The consumer has an ML model $\mathcal{M}$ trained on data $\mathcal{D}_{init} \subset \Gamma$ drawn from $p$. Note that although both $\mathcal{D}_{pool}$ and $\mathcal{D}_{init}$ follow the same distribution $p$, they are not necessarily representative samples of $p$. For example, $\mathcal{D}_{pool}$ may contain a high percentage of records from one part of the data domain $\Gamma$, while $\mathcal{D}_{init}$ from another. In the degenerate case, $\mathcal{M}$ is simply a model randomly initialized without using any training data, i.e., $\mathcal{D}_{init} = \emptyset$. The consumer has a budget $B$, which is the maximum number of records that the consumer can obtain from the provider.

**Query and Predicate.** A query $Q = (P, I)$ consists of a predicate $P$ that specifies the properties of the records the consumer would like to acquire, and an integer $I$ that specifies the number of records requested from the provider. Let $\mathcal{R}_P \subset \mathcal{D}_{pool}$ denote the set of records that satisfy $P$ and $\mathcal{R}_Q$ denote the set of records returned by the provider. All possible predicates admissible to the provider constitute set $\mathcal{P}$, which can be formed in various ways depending on the task at hand. In the work we adopt a simple yet intuitive predicate construction strategy: for a classification problem, $\mathcal{P}$ contains all predicates with a selection on the class label (e.g., `label = 'dog'`); for a regression problem, we discretize each attribute into equal-width sub-ranges, and all combinations of sub-ranges, denoted by "cells", constitute $\mathcal{P}$ (e.g., `1000`$\leq$ `salary` $\leq$ `2000` $\wedge$ `20`$\leq$ `age`$\leq$ `30`). There are many other strategies to construct and refine $\mathcal{P}$. For example, the consumer may perform cross validation on $\mathcal{D}_{init}$ and use the $m$ labels in which $\mathcal{M}$ has the lowest accuracy to construct $\mathcal{P}$ for more targeted acquisition. One may also inject domain knowledge to the predicate construction process and only use these classes or cells related to the task as predicates. For example, a consumer training a cat/dog classifier may not consider predicate `label = 'horse'`. Refer to Section 5.5.1 and Section 5.5.7 for more details on the methodology to construct $\mathcal{P}$ adopted in this work and the associated evaluation. We note that the investigation of predicate construction strategies and their properties is an interesting direction for future work. Our emphasis is to develop methods for data acquisition that are independent of the predicate construction process.

**Interaction.** Each round of interaction between the consumer and the provider consists of two steps: (1) the consumer issues a query $Q = (P, I)$ to the provider, and (2) the provider returns a set of records $\mathcal{R}_Q$, where $|\mathcal{R}_Q| = I$ and each $r \in \mathcal{R}_Q$ is randomly sampled from $\mathcal{R}_P$ [8]. Without loss of generality, we assume that all records provided to the consumer are unique within the same and across different rounds of interactions. A predicate $P$ can be reused across different rounds of interactions as long as $\mathcal{R}_P$ has not been exhausted, i.e., there are records in $\mathcal{R}_P$ that have not been acquired by the consumer yet.

[8]Note that if $I$ is larger than the number of the provider's remaining records (say $I_P$), all of the provider's records will be returned and only $I_P$ will be deducted from the consumer's budget.

**Predicate utility.** After each round of interaction, the consumer estimates the utility of the predicate used in the query, which is useful for planning the next round of interaction. The utility of a predicate $P$ expresses the anticipated accuracy improvement that $\mathcal{R}_P$ brings to $\mathcal{M}$. We define a measure that we call *novelty* to quantify predicate utility. The basic idea of this measure is to quantify the difference between the data acquired in the interaction to those that the consumer currently possesses. The higher the difference, the more information this interaction brings to the consumer. Let $\mathcal{R}_{\mathcal{M}:P}$ be the records the consumer currently possesses satisfying $P$. The novelty of predicate $P$, denoted as $U_P$, is defined on $\mathcal{R}_{\mathcal{M}:P}$ and $\mathcal{R}_P$. More specifically, we consider a binary classification problem that treats $\mathcal{R}_{\mathcal{M}:P}$ and $\mathcal{R}_P$ as samples from class 0 and class 1 respectively, and train a classifier CLF to distinguish between the two sets of records. The utility of $P$ is computed as follows:

$$U_P = \frac{\sum_{(x,y)\in\mathcal{R}_P} \mathbb{I}[\text{CLF}((x,y)) = 1]}{|\mathcal{R}_P|} \tag{5.1}$$

where $\mathbb{I}[*]$ is the indicator function that takes value 1 if statement $*$ is true and 0 otherwise, and $\text{CLF}((x,y))$ denotes the prediction for record $(x,y)$ made by CLF. In principle, any classifier may be used as the CLF. However, in practice it is preferred to use light-weight models for faster training and inference as the computation of novelty is carried out frequently. We study the influence of different classifiers in Section 5.5.9.

The intuition for the design of novelty is that, if $\mathcal{R}_P$ is drawn from a distribution that is very different than the one $\mathcal{R}_{\mathcal{M}:P}$ is drawn from, then the two sets of records can be easily differentiated and $U_P$ is high, and vice versa. Note that we only evaluate the accuracy of the classifier on $\mathcal{R}_P$, because novelty measures how different $\mathcal{R}_P$ is, given $\mathcal{R}_{\mathcal{M}:P}$, rather than the other way around. Such methods for quantifying the difference between two distributions are well adopted in the ML literature [106]. In practice, it may not be feasible for the consumer to obtain all the records in $\mathcal{R}_P$ due to budget limitations. As such, in the proposed solutions, we utilize queries based on the same predicate $P$ returning $|\mathcal{R}_Q| \ll |\mathcal{R}_P|$ records, to estimate $U_P$.

**Acquisition plan.** The acquisition plan of the consumer consists of a sequence of interactions, $\langle (P_1, I_1), (P_2, I_2), \cdots, (P_z, I_z)\rangle$, where $\forall i \in [1, z]$, $P_i \in \mathcal{P}$ and $\sum_{i=1}^{z} I_i = B$. The

consumer receives $B$ records in total after executing the acquisition plan, denoted as $\mathcal{D}_{otd}$.

The problem of data acquisition for model improvement is defined as follows.

**Definition 6.** ***Data Acquisition for Model Improvement****. Given (1) a set of records, $\mathcal{D}_{pool}$ from a provider, (2) an initial set of records, $\mathcal{D}_{init}$, possessed by a consumer, (3) the set of possible predicates, $\mathcal{P}$, (4) the initial model, $\mathcal{M}$, of the consumer, (5) a measure to evaluate model accuracy, $F$, and (6) the budget, $B$, the objective of data acquisition for model improvement is to construct an acquisition plan to maximize $F(M'; \mathcal{D}_{test})$, where $M'$ denotes the consumer model $\mathcal{M}$ after being re-trained on $\mathcal{D}_{init} \cup \mathcal{D}_{otd}$.*

## 5.3 An Estimation-and-Allocation Solution

In this section, we introduce *estimation-and-allocation* (EA), a two-stage solution, to generate the acquisition plan. Essentially, stage one of EA is designed to explore, i.e., to gather more information on how useful each predicate is; while stage two is to exploit, i.e., to utilize the knowledge gained in stage one to optimize subsequent actions. Specifically, the first stage, called the *Estimation Stage*, aims to obtain accurate estimates on the utilities of predicates in $\mathcal{P}$; this is achieved via querying the provider requesting a number of records that constitute a small portion of the budget. Then in the second stage, called the *Allocation Stage*, the consumer allocates the remaining budget and issues queries to the provider based on the estimated predicate utilities. We first discuss in Section 5.3.1 how to ensure the quality of the utility estimates in the Estimation Stage, and present a method that could balance between quality and budget consumption (the amount of record budget spent) in Section 5.3.2. We then elaborate on the Allocation Stage in Section 5.3.3.

### 5.3.1 Estimating Predicate Utility

We now discuss how to estimate the predicate utilities. Recall that the utility $U_P$ of a predicate $P$ is defined as the accuracy of the classifier (denoted by CLF) in differentiating $\mathcal{R}_P$ from $\mathcal{R}_{\mathcal{M}:P}$. Since it is not possible to obtain the entire $\mathcal{R}_P$, we rely on queries using predicate $P$ to effectively sample from it. We can estimate $U_P$ based on the records already acquired with $P$, say $\hat{\mathcal{R}}_P$, and we use $\hat{U}_P$ to denote the estimated value of $U_P$.

The effectiveness of EA depends largely on the accuracy of the predicate utility estimates; thus we investigate how to statistically bound the estimation error, i.e., the difference between $U_P$ and $\hat{U}_P$. To this end, we aim to find the $\epsilon$-$\delta$ approximations of all predicate utilities, defined as follows.

**Definition 7.** *$\epsilon$-$\delta$ **Approximation**. $\hat{U}_P$ is said to be an $\epsilon$-$\delta$ approximation of $U_P$ if $Pr(|U_P - \hat{U}_P| \geq \epsilon) \leq \delta$, where $\epsilon$ denotes the error bound and $\delta$ denotes the significance level.*

In order to determine whether $\hat{U}_P$ is an $\epsilon$-$\delta$ approximation of $U_P$, we conduct a statistical test with the following null hypothesis:

$$H_0^{(P)} : |U_P - \hat{U}_P| \geq \epsilon \tag{5.2}$$

where $P \in \mathcal{P}$ is an arbitrary predicate. We reject $H_0^{(P)}$ at significance level $\delta$ when the following condition is met:

$$\text{reject } H_0^{(P)} \text{ if } \Pr(|U_P - \hat{U}_P| \geq \epsilon) \leq \delta \tag{5.3}$$

The rejection condition bounds the probability of type I error (false rejection) of the statistical test by $\delta$, and if $H_0^{(P)}$ can be rejected, clearly $\hat{U}_P$ is an $\epsilon$-$\delta$ approximation of $U_P$.

We next discuss how to compute $\Pr(|U_P - \hat{U}_P| \geq \epsilon)$. Note that there are two sources of error in estimating $U_P$: (1) approximating $U_P$ with a subset $\hat{\mathcal{R}}_P \subset \mathcal{R}_P$, and (2) the error incurred by the CLF. In our work, we focus on the error caused by insufficient records (i.e., source (1)), and we bound the model error (i.e., source (2)) following [39]. Nonetheless, both types of error can be reduced by increasing the size of $\hat{\mathcal{R}}_P$ [39]. To bound the error attributed to insufficient records, we present the following result on the distribution of $U_P - \hat{U}_P$.

**Theorem 5.** *$U_P - \hat{U}_P \sim \mathcal{N}(0, \frac{U_P(1-U_P)}{|\hat{\mathcal{R}}_P|})$, where $\mathcal{N}(0, \frac{U_P(1-U_P)}{|\hat{\mathcal{R}}_P|})$ denotes the normal distribution with mean 0 and variance $\frac{U_P(1-U_P)}{|\hat{\mathcal{R}}_P|}$.*

*Proof.* Since $U_P$ is the accuracy of the binary classifier CLF in discriminating $\mathcal{R}_P$ from $\mathcal{R}_{\mathcal{M}:P}$, for each record $(x_i, y_i) \in \mathcal{R}_P$, either $\text{CLF}((x_i, y_i))= 0$ or $\text{CLF}((x_i, y_i))= 1$, corresponding to the case when $(x_i, y_i)$ is regarded by CLF to be from $\mathcal{R}_P$ or $\mathcal{R}_{\mathcal{M}:P}$, respectively. Also, if we let $r_i = \mathbb{I}[\text{CLF}((x_i, y_i)) = 0]$, $r_i$ can be viewed as an independent Bernoulli($p$) variable, with $p$ being the probability of $r_i = 1$. Evidently, in this case, $p = U_P$.

As $\hat{U}_P = \frac{1}{|\hat{\mathcal{R}}_P|} \sum_{(x_i,y_i)\in\hat{\mathcal{R}}_P} r_i$ and $r_i \sim$ Bernoulli$(U_P)$, we know $\hat{U}_P * |\hat{\mathcal{R}}_P| \sim$ Binomial$(|\hat{\mathcal{R}}_P|, U_P)$. According to the Central Limit Theorem, we have $U_P - \hat{U}_P \sim \mathcal{N}(0, \frac{U_P(1-U_P)}{|\hat{\mathcal{R}}_P|})$. □

Since $U_P(1-U_P)$ is not known, we cannot directly estimate the difference between $U_P$ and $\hat{U}_P$ based on Theorem 5. Following the standard practice [7] in estimating population mean ($U_P$ in our case), we introduce statistic $t_P$ as follows:

$$t_P = \frac{U_P - \hat{U}_P}{S_P/\sqrt{|\hat{\mathcal{R}}_P|}} \tag{5.4}$$

where $S_P = \sqrt{\hat{U}_P(1-\hat{U}_P)}$ denotes the sample standard deviation. Clearly $t_P$ follows $t$-distribution with degree of freedom $(|\hat{\mathcal{R}}_P| - 1)$.

Now we can re-write $U_P - \hat{U}_P$ as follows:

$$U_P - \hat{U}_P = t_P * S_P/\sqrt{|\hat{\mathcal{R}}_P|} \tag{5.5}$$

The probability of $|U_P - \hat{U}_P| \geq \epsilon$ can now be computed as follows:

$$\begin{aligned}
\Pr(|U_P - \hat{U}_P| \geq \epsilon) &= \Pr\left(|t_P * S_P/\sqrt{|\hat{\mathcal{R}}_P||} \geq \epsilon\right) \\
&= \Pr\left(|t_P| \geq \epsilon\sqrt{|\hat{\mathcal{R}}_P|}/S_P\right) \\
&\leq \int_{-\infty}^{-\frac{\epsilon\sqrt{|\hat{\mathcal{R}}_P|}}{S_P}} f_{n_P}(t_P)dt_P + \int_{\frac{\epsilon\sqrt{|\hat{\mathcal{R}}_P|}}{S_P}}^{\infty} f_{n_P}(t_P)dt_P \\
&= Z_P
\end{aligned} \tag{5.6}$$

where $n_P = |\hat{\mathcal{R}}_P| - 1$, and $f_{n_P}$ is the probability density function of the $t$-distribution with degree of freedom $n_P$.

Therefore, $\Pr(|U_P - \hat{U}_P| \geq \epsilon) = Z_P$, and $H_0^{(P)}$ can be rejected if $Z_P \leq \delta$. Evidently $Z_P$ is negatively correlated to $|\hat{\mathcal{R}}_P|$, and thus if $H_0^{(P)}$ cannot be rejected, one can obtain more records utilizing predicate $P$ and issuing additional queries to reduce the value of $Z_P$ until $Z_P \leq \delta$. The intuition behind this process is that, the more records the consumer obtains utilizing $P$, the more accurate the estimate $\hat{U}_P$ is, and the smaller $|U_P - \hat{U}_P|$ would be. When the null hypotheses for all predicates in $\mathcal{P}$ can be rejected, the current predicate utility estimates are $\epsilon$-$\delta$ approximations.

### 5.3.2 Budget-Aware Utility Estimation

The Estimation Stage has to consider two conflicting goals: providing more accurate estimation of predicate utilities and controlling the budget consumption so that there is more budget left to spend in the Allocation Stage. We thus introduce a budget-aware estimation method, which first acquires a small number of records using each predicate in $\mathcal{P}$ and computes utility estimates accordingly, and then iteratively determines for each predicate whether obtaining more records to improve the estimation accuracy is worthwhile based on a measure called *heuristic reward*. Such a measure is designed to strike a balance between estimation accuracy and budget consumption.

Since for a given significance level $\delta$, the estimation accuracy is contingent on $\epsilon$, we adaptively choose and adjust its value in order to yield estimates with different levels of accuracy. Intuitively, to achieve higher estimation accuracy, i.e., smaller $\epsilon$, the consumer needs to acquire more records for estimation. Assume that the consumer has acquired records $\hat{\mathcal{R}}_P^0$ for each predicate $P \in \mathcal{P}$. We use $B' = B - \sum_{P \in \mathcal{P}} |\hat{\mathcal{R}}_P^0|$ to denote the remaining budget of the consumer. Let $\epsilon_0$ be the minimal $\epsilon$ that causes the null hypotheses for all predicates in $\mathcal{P}$ to be rejected (note that such $\epsilon_0$ always exists, with the extreme case being $\epsilon_0 = 1$). The heuristic reward is defined as $B' \cdot (1 - \epsilon_0)$, which is larger if (1) $B'$ is large, meaning the consumer has more available budget, and (2) $\epsilon_0$ is small, meaning that the estimations are accurate.

**Example 5.3.1.** *Consider a consumer with budget=500 and there are 5 predicates to choose from. Assume that the consumer has acquired 5 records for each predicate, and the resulting sample standard deviations are* $[0.1, 0.11, 0.12, 0.13, 0.14]$ *respectively. Let the significance level* $\delta$ *be 0.01. Using Equation (5.6), we know the minimal values of* $\epsilon$ *causing all* $H_0^{(P)}$ *(*$P \in \mathcal{P}$*) to be rejected are* $[0.21, 0.23, 0.25, 0.27, 0.29]$*, and thus the* $\epsilon$ *that causes all null hypotheses to be rejected, or* $\epsilon_0$*, is 0.29. Since the remaining budget is 475, the heuristic reward is thus* $475 \cdot (1 - 0.29) = 337.25$*.*

Now assume the consumer aims to determine whether reducing $\epsilon_0$ to $\epsilon_b$, by acquiring more records, would improve the heuristic reward. In order to compute the heuristic reward corresponding to $\epsilon_b$, we need to estimate how many additional records need to be acquired,

$\Delta B_b$, to reach $\epsilon_b$. We next show how to estimate the value of $\Delta B_b$.

Recall from Equation (5.6) that the reject condition of $H_0^{(P)}$ is $Z_P \leq \delta$. Since $|\hat{\mathcal{R}}_P|$ is negatively correlated to $Z_P$, in order to get the minimal number of records to acquire to reach a given $\epsilon$, we use the maximal $Z_P$, i.e., $Z_P = \delta$, and rewrite Equation (5.6) as follows:

$$\begin{aligned} Z_P = \delta &\Leftrightarrow \int_{-\infty}^{-\frac{\epsilon\sqrt{|\hat{\mathcal{R}}_P|}}{S_P}} f_{n_P}(t_P)dt_P + \int_{\frac{\epsilon\sqrt{|\hat{\mathcal{R}}_P|}}{S_P}}^{\infty} f_{n_P}(t_P)dt_P = \delta \\ &\Leftrightarrow \int_{-\infty}^{-\frac{\epsilon\sqrt{|\hat{\mathcal{R}}_P|}}{S_P}} f_{n_P}(t_P)dt_P = \frac{\delta}{2} \end{aligned}$$

Let $A_P = \frac{\epsilon\sqrt{|\hat{\mathcal{R}}_P|}}{S_P}$, we can further rewrite the equation above as follows:

$$\int_{-\infty}^{-A_P} f_{n_P}(t_P)dt_P = \frac{\delta}{2} \Leftrightarrow A_P = -\text{PCT}_{n_P}(\frac{\delta}{2}) \tag{5.7}$$

where $\text{PCT}_{n_P}$ denotes the percentile function of $t$-distribution with degree of freedom $n_P$.

Now having a way to determine the value of $A_P$ using Equation (5.7), we rewrite $A_P = \frac{\epsilon\sqrt{|\hat{\mathcal{R}}_P|}}{S_P}$ as follows:

$$|\hat{\mathcal{R}}_P| = \left(\frac{A_P \cdot S_P}{\epsilon}\right)^2 \tag{5.8}$$

Equation (5.8) establishes the relation between the number of records currently obtained and the error bound $\epsilon$ that can be obtained using those records.

Let $\hat{\mathcal{R}}_P^b$ be the records with which we can reach error bound $\epsilon_b$, we have:

$$|\hat{\mathcal{R}}_P^b| = \left(\frac{A_P^b \cdot S_P^b}{\epsilon_b}\right)^2 \tag{5.9}$$

where $S_P^b$ denotes the sample standard deviation of $\hat{\mathcal{R}}_P^b$, and $A_P^b = -\text{PCT}_{n_P^b}(\frac{\delta}{2})$, $n_P^b = |\hat{\mathcal{R}}_P^b| - 1$. Notice that $S_P$ asymptotically converges to the population standard deviation, and $A_P^b$, which is determined by $\text{PCT}_{n_P}$, asymptotically converges to the opposite value of the $\frac{\delta}{2}$-percentile of standard normal distribution [46]. Thus, although $S_P^b$ and $A_P^b$ are unknown, we choose to approximate their values by $S_P^0$ and $A_P^0$, i.e., the values computed based on $\hat{\mathcal{R}}_P^0$, as long as $|\hat{\mathcal{R}}_P^0|$ is reasonably large. We demonstrate the validity of this approximation empirically as well in Section 5.5.2. Therefore, we use $\left(\frac{A_P^0 \cdot S_P^0}{\epsilon_b}\right)^2$ as the least

number of records required to achieve estimation error $\epsilon_b$ for predicate $P$.

**Example 5.3.2.** *Following Example 5.3.1, we assume that the consumer plans to reduce the error bound to 0.15 from 0.29. The values of $A_P^0$ for each $P$ (Equation (5.7)) are:* $[4.68, 4.66, 4.64, 4.63, 4.62]$. *Thus the number of records required for each predicate to reach error bound 0.15 are:* $[10, 12, 14, 17, 19]$.

Since we already posses $|\hat{\mathcal{R}}_P^0|$ records satisfying $P$, we need to acquire $\left(\frac{A_P^0 \cdot S_P^0}{\epsilon^b}\right)^2 - |\hat{\mathcal{R}}_P^0|$ more records utilizing predicate $P$. The total additional records from our budget needed to reach $\epsilon_b$ from $\epsilon_0$ is thus

$$\Delta B_b = \sum_{P \in \mathcal{P}} \left[ \left( \frac{A_P^0 \cdot S_P^0}{\epsilon_b} \right)^2 - |\hat{\mathcal{R}}_P^0| \right]. \tag{5.10}$$

The new heuristic reward after obtaining these $\Delta B_b$ records can thus be estimated as $(B' - \Delta B_b) \cdot (1 - \epsilon_b)$. Let $\epsilon_* = \arg\max_{\epsilon_b}(B' - \Delta B_b) \cdot (1 - \epsilon_b)$, be the value $\epsilon_b$ yielding the maximal heuristic reward. The consumer then compares $B' \cdot (1 - \epsilon_0)$ with $(B' - \Delta B_{\epsilon_*}) \cdot (1 - \epsilon_*)$. If the latter value is higher, meaning that the new combination of the estimation error $\epsilon_*$ and remaining budget $(B' - \Delta B_*)$ is better, we initiate a new interaction to obtain $\Delta B_*$ more records according to Equation (5.10). This process continues until the above calculation indicates no more improvement in heuristic reward is possible via further interaction; we then terminate the Estimation Stage and enter the Allocation Stage.

### 5.3.3 Budget Allocation

The Allocation Stage of EA is concerned with distributing the remaining budget across all predicates in $\mathcal{P}$ utilizing their estimated utilities. We consider two budget allocation strategies, where $M_P$ denotes the budget (number of records) allocated to a predicate $P$:

- **Linear Allocation**: $M_P = \frac{B * \hat{U}_P}{\sum_{P' \in \mathcal{P}} \hat{U}'_P} - B_P$, where $B_P$ denotes the number of records obtained with $P$ during the Estimation Stage.
- **Square-root Allocation**: $M_P = \frac{B * \sqrt{\hat{U}_P}}{\sum_{P' \in \mathcal{P}} \sqrt{\hat{U}'_P}} - B_P$.

We sort all predicates in descending order of their utilities and sequentially obtain records based on $M_P$ starting from the predicate with the highest estimated utility. This stage continues iteratively until the budget is exhausted. We demonstrate in Section 5.5.4 that

each allocation strategy has its own winning cases and therefore the allocation strategy can be selected based on the specific task.

### 5.3.4 The EA Algorithm

The EA solution is summarized in Algorithm 5. We first use $l\%$ ($l$ is configurable) of the budget to acquire records using each predicate to start the estimation (Line 1). We calculate the current estimation quality $\epsilon_0$ (Line 4) and the estimation quality that is expected to yield the highest heuristic reward $\epsilon_*$ (Line 5). The current heuristic reward is compared with the expected highest heuristic reward (Line 7), and if the former is higher, we terminate the Estimation Stage (Lines 7-8) and enter the Allocation Stage (Lines 12-13); otherwise we obtain more records based on $\epsilon_*$ (Lines 10-11) and repeat the process. Note that the Estimation Stage also terminates when the budget is exhausted. However although Line 3 provides an exit to the estimation stage, corresponding to the case when all budget is consumed during estimation, such case will never happen as exhausting all budget provides a heuristic reward of 0 and thus is prohibited by Line 7.

Since during the estimation stage more records can be acquired, it is suggested to initialize Algorithm 5 with a small value of $l$, as initialization with a large value of $l$ may consume too much budget. However, if $l$ is too small (say only 1 record for each predicate), the sample standard deviation $S_P$ computed may accidentally be zero and as a result, $Z_P$ is zero too (Equation (5.6)). Consequently, $H_o^{(P)}$ can be rejected with any $\epsilon$ (even zero) because $Z_P \leq \delta$ is always true, and the estimation stage terminates immediately and abnormally as $\epsilon$ cannot be further reduced. With these trade-offs in mind, we experimentally study the influence of the choice of $l$ in Section 5.5.2.

## 5.4 A Sequential Predicate Selection Solution

In this section, we adopt a Bayesian probabilistic approach and introduce an alternative solution called *Sequential Predicate Selection* (SPS). While EA employs two separate stages for exploration and exploitation, SPS utilizes many rounds of interactions, acquiring a small number of records in each interaction with varying predicates, achieving both exploration

**Algorithm 5** Estimation-and-Allocation

**Input:** budget $B$, all predicates $\mathcal{P}$

**Initialization:** for each $P \in \mathcal{P}$, acquire $l\%$ random records in $\mathcal{R}_P$; assume remaining budget is $B'$.
//Estimation Stage
**while** $B' > 0$ **do**
    $\epsilon_0 \leftarrow$ minimal $\epsilon$ to cause the hypotheses to be rejected;
    $\epsilon_* \leftarrow \arg\max_{\epsilon_b, \epsilon_b < \epsilon_0} (B' - \Delta B_{\epsilon_b}) \cdot (1 - \epsilon_b)$;
    $//\Delta B_{\epsilon_b}$ is computed with Equation (5.10)
    **if** $B' \cdot (1 - \epsilon_0) \geq (B' - \Delta B_{\epsilon_*}) \cdot (1 - \epsilon_*)$ **then**
        break;
    **else**
        acquire $\Delta B'_{\epsilon_b}$ more records according to Equation (5.10);
        $B' = B' - \Delta B_{\epsilon_b}$;
    **end if**
**end while**
//Allocation Stage
Allocate the remaining budget using strategies in Section 5.3.3;

and exploitation in the same round. In particular, in each round, SPS balances between two objectives: (1) acquiring more records using predicates that are expected to provide higher accuracy improvement to the model, based on previous observations; and (2) exploring to identify other predicates that may bring even higher accuracy improvement to model accuracy. We implement the design utilizing Thompson Sampling (TS), an action selection method with proven performance [139, 69].

### 5.4.1 Framework of Thompson Sampling

Thompson Sampling [139] proceeds as follows. Given an action space $\mathcal{A}$, an agent conducts actions $a_1, a_2, \cdots$, each selected from $\mathcal{A}$, in rounds. After applying $a_t$ in round $t$, the agent observes a reward $r_t$, which is randomly generated according to a conditional probability measure $q_\theta(\cdot|a_t)$. The agent is initially uncertain about the value of parameters $\theta$ and thus uses a prior distribution $p$ to describe $\theta$, which is iteratively updated based on $(a_t, r_t)$ pairs. The target is to maximize the cumulative rewards over a given number of rounds. Given $\mathcal{A}$, $p$, and $q$, TS repeats the following steps for action selection:

1. Sample $\hat{\theta} \sim p$, i.e., sample the parameters controlling the reward according to $p$ ($p$ is

called the prior distribution of $\theta$ in this round).

2. Let $a_t = \arg\max_{a \in \mathcal{A}} \mathbb{E}_{q_{\hat{\theta}}}[r_t | a_t = a]$, i.e., $a_t$ is the action with the maximum expected reward under parameter values $\hat{\theta}$. Perform action $a_t$ and observe $r_t$.

3. Update $p = \mathbb{P}_{p,q}(\theta \in \cdot | a_t, r_t)$, i.e., $p$ becomes the posterior distribution of $\theta$ given $(a_t, r_t)$.

In the following, we apply TS to the problem of predicate selection, discuss the choices of $p$ and $q$ for the problem, and develop the update method of $p$ from prior distribution to posterior distribution. In addition, as will be shown in Section 5.4.3, the rewards of predicates evolve over time in our problem, and thus we also design methods to handle changes in $\theta$.

### 5.4.2 Sequential Predicate Selection Using Thompson Sampling

In SPS, we repeatedly issue queries to the provider in multiple rounds of interactions, until the budget $B$ is exhausted. In each round, a query $Q = (P, I)$ is issued and we calculate a value called *query reward* for the records received, deciding on the next query to generate based on the knowledge acquired so far. The objective of the consumer is to maximize the cumulative reward for the queries issued in all rounds of interactions. In what follows, we detail the set of queries that the consumer can issue, how query reward is evaluated, and how knowledge regarding the problem space (queries and the resulting rewards) is updated.

The number of possible queries the consumer may ask is $|\mathcal{P}| \cdot B$, as each pair of $P \in \mathcal{P}$ and $I \in [1, B]$ can form a query. We assume that a fixed $I$, denoted by $I_\Delta$, is used for each query. Thus, choosing a query boils down to selecting a predicate in $\mathcal{P}$, and in the sequel we use the term *predicate reward* of $P$ and the query reward of $Q = (P, I_\Delta)$ interchangeably. We empirically study the influence of $I_\Delta$ in Section 5.5.5.

Let $\mathcal{R}_P^{I_\Delta}$ be the records returned by query $Q = (P, I_\Delta)$. The computation of query reward follows the spirit of predicate utility, i.e., novelty introduced in Section 5.2, except for that instead of using the CLF to differentiate $\mathcal{R}_P$ from $\mathcal{R}_{\mathcal{M}:P}$, the query reward differentiates $\mathcal{R}_P^{I_\Delta}$ from $\mathcal{R}_{\mathcal{M}:P}$, directly measuring the anticipated accuracy improvement $\mathcal{R}_P^{I_\Delta}$ brings to model $\mathcal{M}$. More specifically, the reward of $Q$ is the number of records in $\mathcal{R}_P^{I_\Delta}$ that can be correctly labelled by CLF, following the developments in Section 5.2. Since records in $\mathcal{R}_P^{I_\Delta}$

are randomly selected from $\mathcal{R}_P$, the reward $r$ of $P$ can be viewed as a random variable, which is assumed to follow a distribution $\Pr(r|\theta_P, P)$, where $\theta_P$ refers to the set of parameters of this distribution. In each interaction, the consumer randomly draws a value $r_P$ from $\Pr(r|\theta_P, P)$ for each $P \in \mathcal{P}$, selects the predicate $P^* = \arg\max_{P \in \mathcal{P}} r_P$ and issues query $(P^*, I_\Delta)$.

After obtaining the records for query $(P, I_\Delta)$, we update the distribution $\Pr(r|\theta_P, P)$, by updating the values of $\theta_P$ based on the records received. The distribution $\Pr(r|\theta_P, P)$ before and after the update are the *prior distribution* and *posterior distribution* respectively. We choose to use the Beta distribution [73] to model the prior distribution, which has been shown to be effective in a variety of settings [139]. The Beta distribution is characterized by two parameters $\alpha$ and $\beta$, and it is a particularly good fit for our problem as $\alpha$ and $\beta$ represent the pseudo counts of the number of correct and incorrect classifications we believe the CLF can make, providing our initial perspective of the reward function of $P$. Moreover, as shown in Section 5.3.1, the reward of $(P, I_\Delta)$, i.e., the number of correctly classified records in $\mathcal{R}_P^{I_\Delta}$, follows a Binomial distribution. It is known that the conjugate of Binomial distribution is the Beta distribution [33], and the posterior distribution will still be a Beta distribution, making parameter update highly tractable. We next show how to compute a Beta posterior from a Beta prior and $\mathcal{R}_P^{I_\Delta}$.

Let Beta($\alpha$, $\beta$) be the Beta distribution with two parameters $\alpha$ and $\beta$. In our case, we initialize both $\alpha$ and $\beta$ to 1 for all predicates, essentially making the Beta distribution a uniform distribution, in line with the fact that the consumer has no knowledge regarding the rewards of predicates at the beginning. Suppose that the reward distribution for $P$ is Beta$(\alpha_P, \beta_P)$ before a round of interaction, and $\mathcal{R}_P^{I_\Delta}$ is received in this round. Let $N_P^{I_\Delta}$ be the number of records correctly labelled by CLF, i.e.,

$$N_P^{I_\Delta} = \sum_{(x_i, y_i) \in \mathcal{R}_P^{I_\Delta}} \mathbb{I}[\text{CLF}((x_i, y_i)) = 0] \tag{5.11}$$

We can show that $\alpha_P$ and $\beta_P$ can be updated as follows to obtain the posterior distribution conditional on $N_P^{I_\Delta}$ and $I_\Delta$:

$$(\alpha_P, \beta_P) \leftarrow (\alpha_P, \beta_P) + (N_P^{I_\Delta}, I_\Delta - N_P^{I_\Delta}) \tag{5.12}$$

It follows immediately from Equation (5.12) that (1) the expectation of distribution Beta($\alpha_P, \beta_P$), computed as $\frac{\alpha_P}{\alpha_P+\beta_P}$, is proportional to the reward of $P$ (notice that $\frac{\alpha_P}{\alpha_P+\beta_P}$ is essentially the percentage of records satisfying $P$ that can be correctly labelled by CLF), so that the probabilities of selecting predicates with high observed rewards in future interactions are higher; and (2) after the update, $(\alpha_P + \beta_P + 2)$ is equal to the number of records obtained using $P$ so far (as both $\alpha_P$ and $\beta_P$ are initialized to 1). With more records acquired using $P$, $(\alpha_P + \beta_P)$ becomes larger and the distribution of Beta($\alpha_P, \beta_P$) becomes more concentrated, meaning that we are more confident regarding the expectation of $P$'s reward. Note that this is also the reason why we use the number of correctly labelled records as the reward rather than percentage thereof: even both queries $(P, I_\Delta = 100)$ and $(P, I_\Delta = 10)$ return records of which 80% can be correctly labelled, the former should give us more confidence regarding the distribution of $P$'s reward and thus $(\alpha_P, \beta_P)$ should be greater in this case.

### 5.4.3 Non-stationary Reward Distributions

For our discussion in Section 5.4.2, we have assumed that the reward distribution is stationary regardless of the number of records acquired in previous rounds. However, one can observe that as $\hat{\mathcal{R}}_P$ (the records the consumer has that satisfy predicate $P$) grows, the new information brought by each additional record from $\mathcal{D}_{pool}$ satisfying $P$ decreases, and consequently the reward of $P$ decreases. As such, not all past rewards observed should be treated equally. We should focus on the rewards observed from recent rounds, which better reflect the current reward distributions. Therefore, we modify the posterior computation in Equation (5.12) in a way that remembers only the rewards observed from the most recent $\tau$ rounds of interactions, as inspired by previous research on dealing with non-stationary reward distributions (e.g., [69, 139]).

More specifically, assume that the consumer has interacted with the provider using $P$ for $t$ rounds (including the current round), with rewards $N_P^{I_\Delta}[1], N_P^{I_\Delta}[2], \cdots, N_P^{I_\Delta}[t]$, then $\alpha_P$ and $\beta_P$ are updated as follows in two steps:

$$
\begin{aligned}
&(\alpha_P, \beta_P) \leftarrow (\alpha_P, \beta_P) + (N_P^{I_\Delta}[t], I_\Delta - N_P^{I_\Delta}[t]); \\
&(\alpha_P, \beta_P) \leftarrow (\alpha_P, \beta_P) - (N_P^{I_\Delta}[t-\tau], I_\Delta - N_P^{I_\Delta}[t-\tau]), \text{ only if } t > \tau
\end{aligned}
\tag{5.13}
$$

By remembering only the most recent rewards, the expectation of Beta$(\alpha_P, \beta_P)$ is closer to the current reward of predicate $P$. Besides, "forgetting" the previous rewards prevents Beta$(\alpha_P, \beta_P)$ from becoming too concentrated (recall that $(\alpha + \beta)$ influences how concentrated the distribution is), and thus always allows a chance for more exploration, suitable for the setting with changing rewards.

### 5.4.4 The SPS Algorithm

The operation of SPS is summarized in Algorithm 6. We initialize all reward distributions to Beta$(1, 1)$ (Line 1). At each interaction, we randomly sample a value $r_P$ from distribution Beta$(\alpha_P, \beta_P)$ for each $P$ (Lines 3-4), and select the predicate $P^*$ with the highest $r_P$ and issue query $(P^*, I_\Delta)$ (Lines 5-6). After receiving a set of records, $\mathcal{R}_{P^*}^{I_\Delta}$, we update the values of $\alpha_P$ and $\beta_P$ accordingly (Lines 7-8), merge $\mathcal{R}_{P^*}^{I_\Delta}$ into acquired records and deduct $I_\Delta$ from the remaining budget (Line 9). The process terminates when the budget is exhausted.

**Algorithm 6** Sequential Predicate Selection

**Input:** budget $B$, all predicates $\mathcal{P}$
**Output:** A set of records $\mathcal{R}$
1: **Initialization:** $\forall P \in \mathcal{P}$, $\alpha_P = 1$, $\beta_P = 1$; $\mathcal{R} = \emptyset$
2: **while** $B > 0$ **do**
3: **for** $P$ in $\mathcal{P}$ **do**
4: sample $r_P$ from distribution Beta$(\alpha_P, \beta_P)$;
5: **end for**
6: $P^* = \arg\max_{P \in \mathcal{P}} r_P$;
7: ask query $(P^*, I_\Delta)$ and receive records $\mathcal{R}_{P^*}^{I_\Delta}$;
8: compute $N_{P^*}^{I_\Delta} = \sum_{(x_i, y_i) \in \mathcal{R}_{P^*}^{I_\Delta}} \mathbb{I}[\text{CLF}((x_i, y_i) = 0)]$;
9: update $(\alpha_{P^*}, \beta_{P^*})$ with Equation (5.13);
10: $\mathcal{R} = \mathcal{R} \cup \mathcal{R}_{P^*}^{I_\Delta}$; $B = B - I_\Delta$;
11: **end while**
12: **return** $\mathcal{R}$;

## 5.5 Experiments

The techniques we propose are equally applicable to traditional Machine Learning [19] and Deep Learning [184] models across a variety of domains. We choose models and datasets in the experiments to reflect the wide range of applications we envision. More specifically, we choose state-of-the-art deep models as well as classical ML models to experiment with. The datasets used in the experiments include image data, spatial data, and optical-radar data, and the tasks range from classification to regression.

### 5.5.1 Settings

**Datasets.** We conduct experiments on four datasets. CIFAR10 and CIFAR100 [94] are image classification datasets widely used in the area of ML. The Crop mapping dataset [82] (Crop for short) contains temporal, spectral, textural, and polarimetric attributes for cropland classification. It has 175 real-valued features and one target (seven crop types). 3D Road Network [81] (RoadNet for short) is a geographical dataset consisting of tuples of longitude, latitude, and altitude. Following the instruction in [81], we use longitude and latitude as the features and altitude as the predicted value. The latter two datasets can also be found in the UCI data repository [37]. The characteristics of the four datasets are summarized in Table 5.1.

CIFAR10, CIFAR100, and Crop are used for classification tasks, while RoadNet is used for a regression task. To generate $\mathcal{D}_{test}$, we directly use the test sets provided by CIFAR10 and CIFAR100, and randomly select 20% records from Crop and RoadNet.

Table 5.1: Dataset Characteristics

| *dataset* | *# records* | *# classes* | *# dimensions* |
|---|---|---|---|
| CIFAR10 | 60,000 | 10 | 1,024 |
| CIFAR100 | 60,000 | 100 | 1,024 |
| Crop | 325,834 | 7 | 175 |
| RoadNet | 434,874 | N/A | 2 |

**Models.** For classification on CIFAR10 and CIFAR100, we adopt VGG8B with predsim

loss function [127], one of the most recent and state-of-the-art deep learning structures. For classification on Crop, we use Decision Tree. For RoadNet, we use $k$NN Regressor [3]. We also utilized other applicable models for our evaluation (e.g., AlexNet [95], EfficientNet [158]) and observed similar trends. We use the code of VGG8B provided by the authors, and the Decision Tree and $k$NN Regressor implementation in scikit-learn. Default settings are adopted for all models.

**Construction of $\mathcal{P}$.** For CIFAR10, CIFAR100, and Crop, we build predicates based on class labels, resulting in 10 predicates for CIFAR10, 100 predicates for CIFAR100, and 7 predicates for Crop. For RoadNet, by default we discretize the data space by partitioning the range of each feature into four equal-width sub-ranges, resulting in $4^2 = 16$ cells; a predicate selects records falling into a specific cell. We study the influence of $|\mathcal{P}|$ in Section 5.5.7.

**Construction of $\mathcal{D}_{init}$.** We construct $\mathcal{D}_{init}$ using 20% records in the corresponding dataset $\mathcal{D}$ for CIFAR10 and CIFAR100, and 1% records for Crop and RoadNet. We select records from $\mathcal{R}_P$ for each $P \in \mathcal{P}$ to construct $\mathcal{D}_{init}$ following a power-law distribution. More specifically, with a random order of predicates in $\mathcal{P}$, let $P_i$ be the $i$-th predicate ($i \in [1, |\mathcal{P}|]$), the number of records selected from $\mathcal{R}_{P_i}$ is proportional to $i$.

**Selection of CLF.** We use the $k$NN classifier with $k = 1$ as the CLF in our experiments. We study the impact of varying $k$ values as well as utilizing diverse classifiers in Section 5.5.9. For Crop and RoadNet, we directly use the raw attributes as the input of CLF. In accordance to previous work on image-based $k$NN search [168], for CIFAR10 and CIFAR100, we first use HOG [24] (Histogram of Oriented Gradients), a widely-adopted image feature extractor, to transform an image into a feature vector, and use the transformed feature vectors as the input of CLF.

**Evaluation.**[9] Let $\mathcal{D}_{otd}$ be the records acquired during the acquisition process. We train the model on $\mathcal{D}_{init} \cup \mathcal{D}_{otd}$ and evaluate on $\mathcal{D}_{test}$. For the classification tasks on CIFAR10, CIFAR100, and Crop, accuracy is computed as follows:

$$accuracy = \frac{\sum_{(x,y)\in\mathcal{D}_{test}} \mathbb{I}[\mathcal{M}(x) = y]}{|\mathcal{D}_{test}|} \tag{5.14}$$

where $\mathcal{M}(x)$ denotes the model output on $x$.

[9] code available at: https://github.com/AwesomeYifan/Data-acquisition-for-ML

For the regression task on RoadNet, we use $R^2$ score to evaluate the performance, computed as follows:

$$R^2 = 1 - \frac{\sum_{(x,y)\in\mathcal{D}_{test}}(y - \mathcal{M}(x))^2}{\sum_{(x,y)\in\mathcal{D}_{test}}(y - \bar{y})^2} \tag{5.15}$$

where $\bar{y} = \frac{\sum_{(x,y)\in\mathcal{D}_{test}} y}{|\mathcal{D}_{test}|}$.

The acquisition process and model training are repeated ten times and the average accuracy/$R^2$ score is reported.

### 5.5.2 The Effect of $l$ on EA

As described in Section 5.3, EA requires acquiring $l\%$ random records for each predicate from the provider for initialization. Here, we experimentally evaluate the effect of $l$ on the performance of EA. To have a common basis for the evaluation of the trends we fix the significance level ($\delta$) at 0.001 throughout the experiment. The results are provided in Figure 5.1.

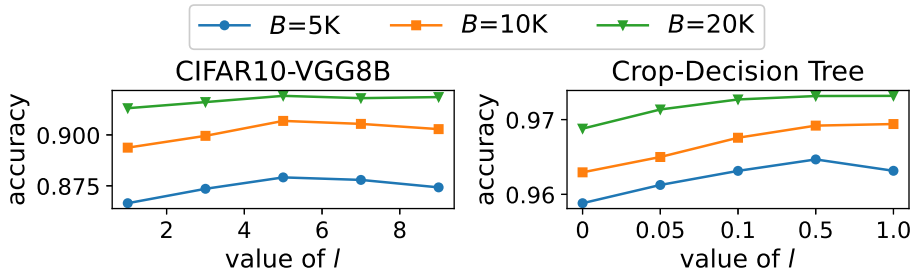



Figure 5.1: Effect of $l$ on EA

As demonstrated in Figure 5.1, the performance of EA with a small budget is more sensitive to the choice of $l$, while $l$ has a less significant impact on EA's performance when a large budget is used, except for cases where $l$ is very small. The trade-off involved in selecting $l$ can be summarized as follows. Using an overly-large $l$ would result in too much budget consumption for the initialization, and consequently reduce the budget available for the Allocation Stage of EA, especially when the total budget $B$ is small. On the other hand, using too small an $l$ may cause the Estimation Stage to perform badly, leading to low-quality predicate utility estimates, as discussed in Section 5.3.4. In the following experiments, we

set $l = 5$ for CIFAR10 and CIFAR100, and $l = 0.5$ for RoadNet and Crop. Since the budgets we use are fairly large for the respective dataset, the performance of EA is less dependent on the choice of $l$.

**Takeaways.** (1) An overly small or large $l$ may harm the performance of EA. The value $l = 5$ for image data, and $l = 0.5$ for spatial data worked best during our experiments; (2) The performance of EA becomes less dependent on the choice of $l$ as budget increases.

### 5.5.3 Estimation Accuracy of EA

In the Estimation Stage of EA, we assess the utility of predicate $P$, $U_P$, based on the records acquired with $P$, $\hat{\mathcal{R}}_P$. Let $\hat{U}_P$ be the estimated value of $U_P$. In this section, we examine the estimation accuracy. More specifically, we vary the number of records used for utility estimation, normalized by $|\mathcal{R}_P|$, i.e., $|\hat{\mathcal{R}}_P|/|\mathcal{R}_P|$, to study its effect on the absolute error of the estimation, i.e., $|U_P - \hat{U}_P|$. The results are presented in Figure 5.2.

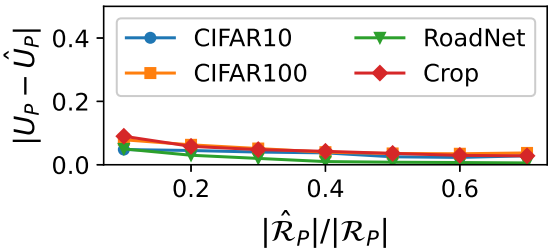



Figure 5.2: Estimation Accuracy

As can be observed from Figure 5.2, the absolute error of the estimation is consistently low (below 0.1), and increasing the number of records used for estimation further reduces the absolute error. The reason is that, according to Theorem 5, $U_P - \hat{U}_P$ follows a normal distribution, and increasing $|\hat{\mathcal{R}}_P|$ reduces the standard deviation of the distribution; consequently, the value of $\hat{U}_P$ converges to the true utility, $U_P$.

### 5.5.4 Comparison of Allocation Strategies in EA

Two strategies have been proposed in Section 5.3.3, namely, Linear Allocation and Square-root Allocation, which allocate the budget proportional to the estimated utility of each

| $B$ | $H_A$ | p-value |
|---|---|---|
| 5,000 | $\mu_L > \mu_S$ | 4e-3 |
| 20,000 | $\mu_S > \mu_L$ | 1e-7 |

Table 5.2: Significance Test

predicate or its square-root respectively. In this section we evaluate the impact of the allocation strategy on the performance of EA, and present the results in Figure 5.3. We also conduct one-tailed t-tests to determine whether the average accuracy improvement of one allocation strategy is statistically higher than the other. The cases for $B = 5,000$ and $B = 20,000$ are provided in Table 5.2, where $\mu_L$ ($\mu_S$) denotes the average accuracy improvement of Linear (Square-root) Allocation.

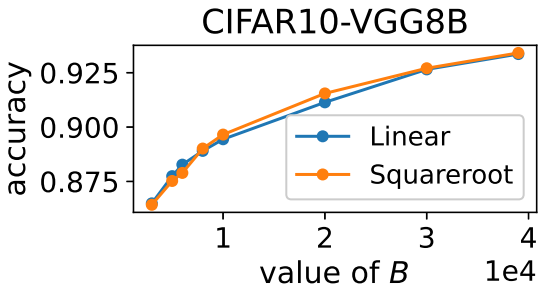



Figure 5.3: Allocation Strategies

As is clear from Figure 5.3 and Table 5.2, different allocation strategies achieve similar performance overall, and with significance level $\alpha < 0.01$ we say Linear Allocation yields higher accuracy when the budget is relatively small (e.g., $B = 5,000$) and Square-root Allocation takes the lead when the budget is large (e.g., $B = 20,000$). This observation is the result of two competing underlying factors: on one hand, we should exploit the knowledge on the predicate utility gained from the Estimation Stage and therefore we should bias the allocation towards predicates with higher estimated utilities as much as possible (hence the superiority of Linear Allocation over Square-root Allocation for small budgets); on the other hand, as discussed in Section 5.4, the utility of a predicate $P$ decreases as more records are acquired with $P$, i.e., the marginal benefit of obtaining records using predicates with higher estimated utilities becomes lower as the budget grows (hence Square-root Allocation outperforms Linear Allocation for large budgets). In other experiments we adopt Linear

Allocation.

**Takeaways.** (1) Linear Allocation is better suited for data acquisition with budget $B \leq 0.2|\mathcal{D}|$ in our experiments; (2) Square-root Allocation is better suited for data acquisition with budget $B > 0.2|\mathcal{D}|$ during our evaluation.

### 5.5.5 The Effect of Batch Size on SPS

With SPS, records are acquired in small batches of size $I_\Delta$. We evaluate the effect of $I_\Delta$ on the performance of SPS presenting our results in Figure 5.4.

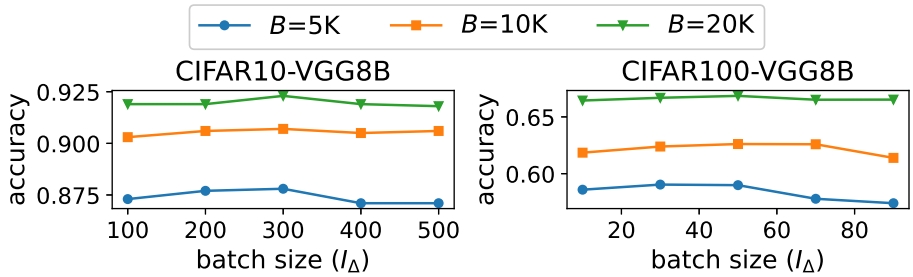



Figure 5.4: Effect of Batch Size on SPS

As can be observed from Figure 5.4, data acquisition with a small budget is more sensitive to the value of $I_\Delta$. The trade-off in selecting $I_\Delta$ is as follows. Let $(P, I_\Delta)$ be the query. With a small $I_\Delta$, the records returned by the provider may not be representative of $\mathcal{R}_P$, and the query reward thus computed is inaccurate. Consequently SPS cannot identify those predicates with high rewards. On the other hand, if $I_\Delta$ is too large, then each interaction consumes too much budget, limiting the exploitation of predicates with higher rewards. However, as the budget increases, SPS becomes progressively less sensitive to the batch size, because (1) the variations in the records obtained in each interaction have minimal impact on the overall estimation accuracy given a large number of interactions; (2) although exploration with large $I_\Delta$ consumes more budget, it also provides more information regarding the reward distribution (see Equation (5.12)), benefiting future predicate selection. In the following experiments, we select $I_\Delta = 300$ for CIFAR10, and $I_\Delta = 30$ for CIFAR100, Crop and RoadNet. Since the budgets we use are fairly large for the respective datasets, we expect less dependency on the choices of $I_\Delta$.

**Takeaways.** (1) The accuracy of SPS is relatively stable over a wide range of batch size values, only slightly decreasing for an overly small or large batch size, depending on the ML/DL model and data; (2) The accuracy of SPS becomes less dependent on the batch size as budget increases.

### 5.5.6 The Effect of $\tau$ on SPS

As discussed in Section 5.4.3, to deal with the non-stationary reward, we only use the records acquired by the most recent $\tau$ queries with $P$ to update the posterior distribution of $P$'s reward. We evaluate the effect of $\tau$ on the performance of SPS, and report the results on CIFAR10 in Figure 5.5; similar trends can be observed on other datasets.

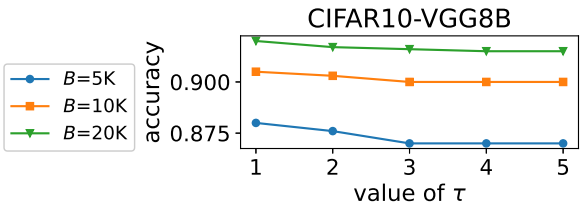



Figure 5.5: Effect of $\tau$ on SPS

As can be observed from Figure 5.5, the performance of SPS is relatively stable across different values of $\tau$, with small $\tau$ yielding slightly higher accuracy. The reason is that, as indicated by Equation (5.13), with larger $\tau$, $\alpha_P$ and $\beta_P$ contain more dated reward observations and thus diverge from the current reward distribution; this in turn may cause the acquisition of less useful records (in terms of accuracy improvement). Having less useful records clearly has a stronger influence on model accuracy when the total number of training records is smaller (corresponding to a smaller budget). However, with the SPS strategy, as long as the predicate with the actual maximal expected reward has a higher probability to be selected than the other predicates, the cumulative reward is likely to be maximized. Therefore, SPS is tolerant to inaccurate reward distribution estimations and robust to the value of $\tau$. We set $\tau$ to 1 in the following experiments.

**Takeaways.** (1) Setting $\tau = 1$ always leads to higher accuracy during our evaluation; (2) SPS is relatively robust with respect to $\tau$.

### 5.5.7 The Effect of $|\mathcal{P}|$

In this section we study the influence of the number of predicates, i.e., $|\mathcal{P}|$, on the accuracy of the methods. More specifically, we change $|\mathcal{P}|$ by changing either: (1) the number of labels to construct $\mathcal{P}$ for a classification dataset, or (2) the discretization granularity for a regression dataset. For case (1), we select CIFAR100 and use the $m$ labels with which the model has the lowest accuracy (cross validated on $\mathcal{D}_{init}$) to construct $\mathcal{P}$. For case (2), we use RoadNet and partition the range of each of the two features into $n$ equal-width sub-ranges, resulting in $n^2$ cells, which are used as $\mathcal{P}$. The results are presented in Figure 5.6.

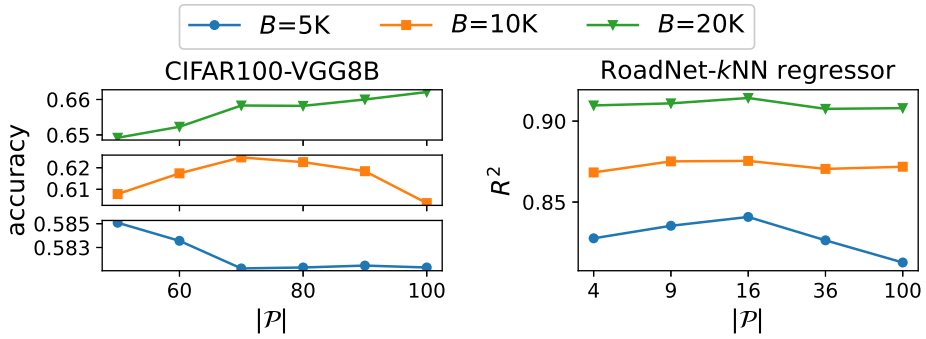



Figure 5.6: Effect of Space Discretization

As is clear from the results on CIFAR100, for case (1), the trend of accuracy with respect to $|\mathcal{P}|$ depends on the budget. The $|\mathcal{P}|$ yielding the maximal accuracy is 50 for $B$=5K, 70 for $B$=10K, and 100 for $B$=20K. The trend confirms our intuition that with a small budget, limiting the data acquisition to the predicates where the model is more error-prone reduces extra exploration cost and consequently increases the accuracy; however, this may result in over-exploitation of these predicates when the budget is large due to decreasing utility (as discussed in Section 5.4.3), leading to a slower increase in accuracy compared to larger $\mathcal{P}$. As can be observed from the results on RoadNet, for case (2), the $R^2$ score is relatively stable with respect to $|\mathcal{P}|$, with a slight increase when $|\mathcal{P}|$ is between 9 and 36. The reason is that, an overly coarse partitioning granularity (small $|\mathcal{P}|$) prevents the effective identification of the area in the data space where the model has a low $R^2$ score, while an overly fine partitioning granularity (large $|\mathcal{P}|$) increases the exploration cost as there are more predicates whose utilities need to be estimated.

| $B$ | $H_A$ | p-value |
|---|---|---|
| 3,000 | $\mu_E > \mu_P$ | 1e-4 |
| 20,000 | $\mu_P > \mu_E$ | 1e-9 |

Table 5.3: Significance Test

**Takeaways.** During our experiments, (1) Descritizing the data space such that each cell occupies $3\% - 10\%$ of the size of the entire data space gives higher $R^2$ score; (2) For classification tasks with budget $B \leq 0.2|\mathcal{D}|$, using no more than 70% of all labels to construct $\mathcal{P}$ gives higher accuracy; with $B > 0.2|\mathcal{D}|$, using at least 70% of all labels to construct $\mathcal{P}$ leads to higher accuracy; (3) The performance of the proposed methods is generally stable in terms of the number of predicates.

### 5.5.8 EA vs. SPS

We now experimentally compare the two methods proposed in the work, EA and SPS, and showcase the relative trends. We report the results in Figure 5.7. We also conduct one-sided t-tests to determine whether the average accuracy improvement of one method is statistically higher than the other. The cases for $B = 3,000$ and $B = 20,000$ are provided in Table 5.3, where $\mu_P$ ($\mu_E$) denotes the average accuracy improvement of SPS (EA).

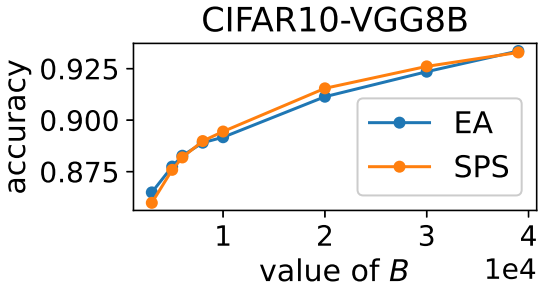



Figure 5.7: EA vs. SPS

As can be observed from Figure 5.7 and Table 5.3, EA and SPS provide similar performance, and with significance level $\alpha < 0.001$ we say EA provides a higher accuracy gain with a small budget (e.g., $B = 3,000$) and SPS provides a higher accuracy improvement with a large budget (e.g., $B = 20,000$). The reason is that EA is a budget-aware method: with a small budget it tends to allocate less budget for utility estimation, aided by the heuristic

reward, so that more budget can be allocated to predicates with high estimated utilities. The mechanism of SPS, on the other hand, is budget-agnostic, and the exploration of all predicate rewards at the start consumes budget and limits the chances to exploit predicates with high utilities, especially when the budget is small. As the budget increases, SPS starts to outperform EA since it acquires records in small batches and can flexibly adjust the acquisition strategy in face of utility changes; EA conducts one-time allocation without considering future utility changes and the records thus obtained may not be impactful to improve model accuracy.

**Takeaways.** In our evaluation, (1) EA is better suited for data acquisition with budget $B \leq 0.2|\mathcal{D}|$; (2) SPS is better suited for data acquisition with budget $B > 0.2|\mathcal{D}|$.

### 5.5.9 The Effect of CLF

As discussed in Section 5.2, the utility of a predicate is essentially the accuracy of a classifier (CLF) in differentiating $\mathcal{R}_P$ and $\mathcal{R}_{\mathcal{M}:P}$. Here we experimentally study the influence of CLF on the accuracy improvement. More specifically, since the utility computation is carried out frequently, we consider lightweight models including the $k$NN classifier ($k \in \{1, 3, 5\}$), the decision tree classifier, and the perceptron classifier. We report results on CIFAR10 in Figure 5.8; similar trends can be observed on other datasets.

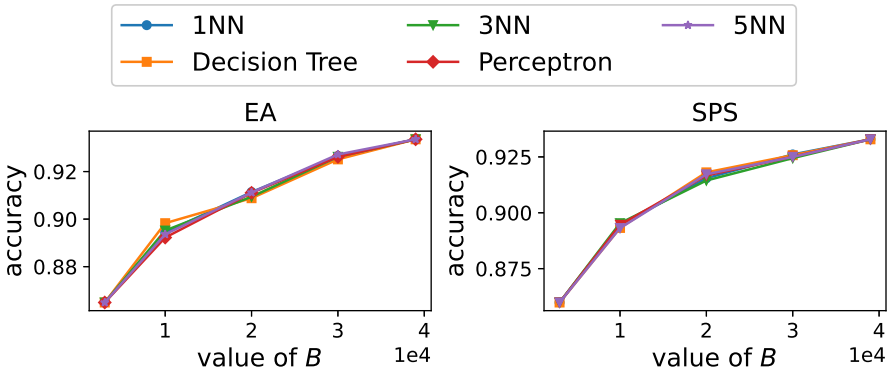



Figure 5.8: Effect of CLF

As can be observed from Figure 5.8, the performance of the methods is not sensitive to the particular model used as the CLF and its hyper-parameters. In other experiments, we

use the $k$NN classifier with $k = 1$.

### 5.5.10 Comparison with Baseline Methods for Data Acquisition

The closest piece of work that can be adapted to our setting for comparison purposes is the work on active class selection (ACS) [105]. Although the problem solved therein is different, we can adapt the methods used for our setting. Therefore, we use ACS (adapted to our setting) as the baseline and present experimental results comparing it with our proposals. We note, however, that this is not a fair comparison, because ACS requires re-training the model after each interaction which can be computationally prohibitive for complex models involving large datasets, whereas ours does not.

Specifically, ACS acquires $b$ new data records in each round (the same batch size as used in Section 5.5.5), which is allocated to each class uniformly (ACS-Uniform), or in proportion with the accuracy improvement for each class (ACS-AI), or the number of records in each class whose label has changed (ACS-RD) during the last round. Note that ACS-AI and ACS-RD require model re-training after new records are obtained and thus are too expensive to be applied to CIFAR10 and CIFAR100. As such, we apply ACS-Uniform to CIFAR10, and ACS-AI and ACS-RD to Crop, with results provided in Figure 5.9. The observations on other datasets are similar.

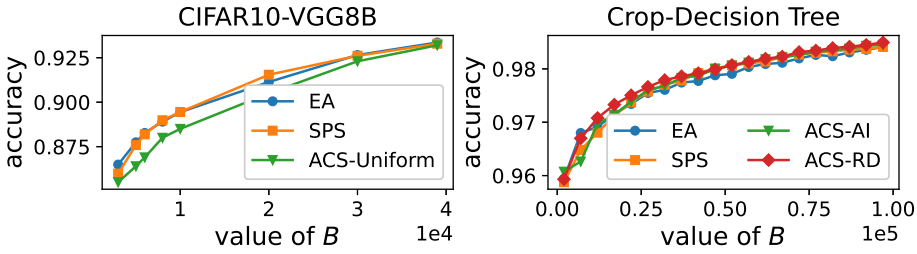



Figure 5.9: Comparison with Baselines

The results in Figure 5.9 (a) indicate that EA and SPS consistently outperform ACS-Uniform (except for the case of $B = 4,000$ when all samples are acquired) and achieve similar accuracy to baselines *that require the model to be re-trained after each interaction*, by effectively acquiring records with higher novelty that are more likely to boost accuracy.

### 5.5.11 Comparison with Utility Measures Based on Re-training/refinement

One of the main advantages of the utility measure we propose, novelty, is that it does not require the computationally expensive step of model re-training. In this section, we compare novelty with re-training-based utility measures in terms of model accuracy improvement and data acquisition cost. More specifically, we compare with a re-training-based measure where $\mathcal{M}$ is re-trained on newly-acquired records, say $\mathcal{R}_P^I$, and the improvement in accuracy after re-training is used as the utility of $P$. While all lightweight models are re-trained from scratch, we adopt the state-of-the-art incremental learning method, UCB [40], to refine deep models instead of conducting complete re-training to keep training overhead manageable. We report the results of using SPS together with both utility measures on CIFAR10 and Crop in Figure 5.10; observations are similar on other datasets and consistent in the case of using EA.

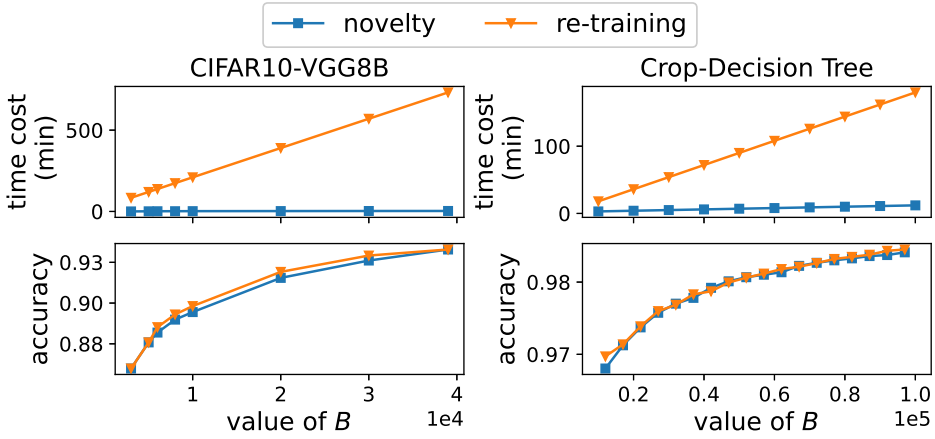



Figure 5.10: Comparison with Re-training-based Measure

The results in Figure 5.10 indicate that novelty achieves similar accuracy to the re-training-based utility measure by effectively acquiring records with higher novelty that are more likely to boost accuracy, while requiring orders of magnitude less time. Although the model can be refined incrementally with UCB, repetitive model refinement, especially when the budget is large, still results in high execution overhead. Although lightweight models such as decision trees can be constructed rapidly, repetitive construction imposes large computational overheads during the acquisition process. Novelty, on the contrary,

conducts data acquisition by only looking at the data, and thus is highly efficient for practical deployments.

## 5.6 Summary and Future Work

In this work, we have considered the problem of acquiring data in order to improve the accuracy of ML models, and laid out the framework of interaction between a provider and a consumer in the context of data markets. We have proposed two algorithmic solutions that the consumer with a limited budget could use to obtain data from the provider, striking a balance between exploration (gaining more knowledge on the data the provider possesses) and exploitation (utilizing that knowledge for allocating the limited budget for data acquisition). The first solution, EA, has two distinctive stages, Estimation Stage and Allocation Stage, focusing on exploration (obtaining accurate estimates on the predicate utilities) and exploitation (allocating the budget according to the estimates) respectively. The second solution, SPS, blends exploration and exploitation in each round of interaction, and adaptively allocates budget for the next round, investing resources into more promising areas of the data space to improve model accuracy. Results from our experimental studies have confirmed the effectiveness of our proposals, and illustrated the trade-offs and relative strengths of each proposed solution.

In our work, we focus on a setting where the consumer has no knowledge regarding the provider's data, which we believe is practical for real-world data markets. To reflect this assumption, the strategies of both EA and SPS initially start from uniformly acquiring data using different predicates. The exploration of different initialization strategies (such as biasing to certain predicates) would be an important direction when we design data acquisition strategies for markets where the consumer has prior knowledge regarding the provider's data distribution.

As an alternate of SPS, we can also adopt methods such as Markov Decision Process (MDP) [11] together with Thompson Sampling to model the change in predicate utilities and assist the predicate selection process, and defining the states of MDP and designing the corresponding data acquisition policy would be interesting future work. In addition, with

SPS we construct reward distributions based on observations from the recent rounds to cater to the nonstationary nature of predicate utilities, and other methods such as discounting the relevance of previous observations are also applicable. The designs of the discounting function when adopting such an variant of SPS in different scenarios as well as the comparison of different nonstationary update methods would be natural extensions of the current work.

# 6 Conclusion and Future Work

## 6.1 Summary of the Thesis

In this thesis we have summarized three of our representative works in the intersection area of data management and Machine Learning. First, a learned indexing and searching framework for exact set similarity search, LES$^3$, which follows a filter-and-verify methodology for efficient query processing. LES$^3$consists of two main structures: TGM partitions data into disjoint sets, balancing between index access cost and effectiveness in pruning candidate sets, and L2P utilizes a cascade of Siamese networks to identify the (near) optimal partitioning of data which leads to the maximal pruning power. Second, leveraging the execution time distributions of query plans to improve the performance of DBMS fulfills the user's various objectives regarding query performance. We enhanced conformal prediction technique to produce execution time distributions for arbitrary plans, which may be integrated into both conventional cost estimators and learned cost estimators, incurring very minor overhead. We have conducted extensive experiments on multiple benchmarks, and the results validated the effectiveness of the proposed method in achieving various objectives regarding query performance. Third, defining and solving the problem of data acquisition with improving the accuracy of a ML model as the objective. We have proposed intuitive and robust measures to quantify how useful a collection of new data is to the ML model, and developed two algorithmic approaches for the ML owner (or data consumer) to acquire the most useful data with a limited budget.

## 6.2 Future Work

The works presented in this thesis represent our first step in contributing to the research in the intersection area of data management and Machine Learning, and many interesting and practical problems remain open. Below we list several representative topics worth further investigation:

**Extension of LES$^3$.** In Section 3 we consider an ideal setting with separable data and minor distribution drifts to develop the learning-based index structure LES$^3$. In order to make the technique more practical, we may consider the following extensions. First, consider overlaps among groups when minimizing $GPO$. While overlap among groups generally makes partitioning-based similarity search more difficult, LES$^3$ can be optimized to more effectively handle such cases. For example, when sets in a particular group are very close to each other, further partitioning the group does not help to improve the pruning efficiency, and thus we can terminate the training corresponding to the group in the cascade training framework, and develop more suitable index methods for this group. Second, deployment of LES$^3$ in an online environment. In an online environment where new sets with possibly different distribution arrive continually, assigning new sets to groups using the pre-trained partitioner may cause significant overlap among groups and decrease the pruning efficiency. Therefore, detecting drifts in data distribution and refining the learned partitioner frugally are of great importance when adopting the technique for practical tasks.

**Extension of dbET.** Further optimization opportunities exist for the technique developed in Section 4. For example, we may consider multi-query optimization when utilizing the plan selection framework introduced in Figure 4.5, i.e., leveraging the overlap or correlation among a batch of queries to select a plan for each query so as to share computation and reduce the overall execution time. In addition, when generating candidate plans for a particular query, we can reduce the execution time distribution construction cost by quantifying the similarity between plans (by means such as computing the distance between embedded plans) and only consider plans with high level of difference to generate a diverse set of candidate plans.

**Extension of data acquisition strategies.** In Section 5 we consider an ideal data

acquisition platform consisting of a single data provider to construct the foundation of the task, in practice, however, the scenario may be more complex, and a natural extension of the problem is to consider multiple data providers with various data coverage, data quality, and prices. In such a setting, the data consumer has more flexibility to choose which type of data to acquire, and which provider(s) to acquire data from, based on the budget limitation. For example, the consumer may need to decide whether to acquire high-quality data with a higher price, or data with fair quality but a lower price is sufficient for the ML model at hand. While finding the optimal acquisition strategy in such a setting is more challenging, the interaction pattern of the setting better describes the practical data market and thus is of great importance in building a healthy and long-running data market.

**Siamese network for plan selection.** While existing query optimizers generally follow the methodology that first estimates the cost of each candidate plan and then selects the one with the minimal cost, various ML techniques, especially Siamese network, provide a new angle to solve the plan selection problem. Siamese network consists of two duplicate models sharing the same set of parameters and an output module on top of the two siamese models, and it is generally used for tasks such as estimating the similarity between a pair of pictures. With necessary modification, Siamese network can also be utilized for plan selection tasks. More specifically, we can provide the network with two encoded plans as the input, and the output indicates whether the first plan is more efficient than the second plan. The advantage of utilizing Siamese network for plan selection is that, compared with cost estimation, the objective of which is to accurately predict the execution time of each plan, the task of directly finding the faster one between two plans seem to be much easier, so that training such a model should be more efficient and the objective can be better satisfied.

**Maintainability of learned query optimizers.** One severe challenge in integrating learned optimizers into real DBMSs is that training the optimizer is usually much more expensive than building conventional cost estimators such as histograms, making it hard to maintain in practice, especially when data updates frequently occur as in OLTP (online transactional processing) scenarios. In order to make the learned query optimizer practical for deployment, we need to answer the following two questions: (1) when to update the model, and (2) how to update the model. Determining when to update the model is important in

reducing the extra training overhead, as it is unnecessary to update the model on each insertion of new records. We can detect the change in data distribution, and only update the model when the change reaches a certain threshold, meaning that the model can no longer make satisfying plan selection choices. Besides, when updating the model, we do not need to construct it again from scratch, as the model has already captured some characteristics of the data. Therefore, we need to design frugal model update approaches for the problem to improve maintainability, including the selection of training samples (e.g., only use plans whose execution times have significantly changed as the training data), and model refinement strategy (e.g., freeze certain layers of the model).

**Integrating learned index into real systems.** A large number of learned indexes have been proposed in the past few years dealing with different tasks and focusing on various scenarios, and they outperform conventional indexes on a variety of benchmarks in terms of both search efficiency and storage overhead. While we have good reason to expect satisfying performance of learned index when integrated into real systems, deploying the technique may arise new challenges. For example, the selection of model for different workload patterns and requirements to balance between training/inference overhead and prediction accuracy, interpreting and debugging the model when unexpected performance occur, etc.

**Acquiring data from multiple providers with various coverage, quality, and price.** In Section 5 we consider an ideal data acquisition platform consisting of a single data provider to construct the foundation of the task, in practice, however, the scenario may be more complex, and a natural extension of the problem is to consider multiple data providers with various data coverage, data quality, and prices. In such a setting, the data consumer has more flexibility to choose which type of data to acquire, and which provider(s) to acquire data from, based on the budget limitation. For example, the consumer may need to decide whether to acquire high-quality data with a higher price, or data with fair quality but a lower price is sufficient for the ML model at hand. While finding the optimal acquisition strategy in such a setting is more challenging, the interaction pattern of the setting better describes the practical data market and thus is of great importance in building a healthy and long-running data market.

Besides the topics detailed above, there are also other interesting and important prob-

lems for future studies, such as learned indexes for fuzzy match with bounded errors, data markets with data brokers. We believe the direction of bridging data management and Machine Learning is fruitful and of great importance for both academic research and industrial applications, and may eventually lead to a smarter world. We will continue working on related topics and we hope the studies conducted in this thesis would encourage researchers with interests to join us and together contribute to this new and very promising area.

# Bibliography


[1] Anish Agarwal, Munther Dahleh, and Tuhin Sarkar. "A marketplace for data: An algorithmic solution". In: *Proceedings of the 2019 ACM Conference on Economics and Computation*. 2019, pp. 701–726.

[2] Pulkit Agrawal et al. "Data Platform for Machine Learning". In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 2019, pp. 1803–1816.

[3] Naomi S Altman. "An introduction to kernel and nearest-neighbor nonparametric regression". In: *The American Statistician* 46.3 (1992), pp. 175–185.

[4] *Amazon Mechanical Turk*. URL: `https://www.mturk.com/`.

[5] *Appen*. URL: `https://appen.com/`.

[6] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. "Efficient exact set-similarity joins". In: *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment. 2006, pp. 918–929.

[7] Robert B Ash et al. *Probability and measure theory*. Academic Press, 2000.

[8] Brian Babcock and Surajit Chaudhuri. "Towards a robust query optimizer: a principled and practical approach". In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 2005, pp. 119–130.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[10] Magdalena Balazinska, Bill Howe, and Dan Suciu. "Data markets in the cloud: An opportunity for the database community". In: *Proceedings of the VLDB Endowment* 4.12 (2011), pp. 1482–1485.

[11] Richard Bellman. "A Markovian decision process". In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[12] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: A methodological tour d'horizon". In: *European Journal of Operational Research* (2020). ISSN: 0377-2217.

[13] Ingwer Borg and Patrick Groenen. "Modern multidimensional scaling: Theory and applications". In: *Journal of Educational Measurement* 40.3 (2003), pp. 277–280.

[14] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. "Spatio-textual similarity joins". In: *Proceedings of the VLDB Endowment* 6.1 (2012), pp. 1–12.

[15] Jane Bromley et al. "Signature verification using a" siamese" time delay neural network". In: *Advances in neural information processing systems*. 1994, pp. 737–744.

[16] Ümit V Çatalyürek and Cevdet Aykanat. "Patoh (partitioning tool for hypergraphs)". In: *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1479–1487.

[17] Shuchi Chawla et al. "Revenue Maximization for Query Pricing". In: *Proc. VLDB Endow.* 13.1 (2019), pp. 1–14.

[18] Shuchi Chawla et al. "Revenue maximization for query pricing". In: *arXiv preprint arXiv:1909.00845* (2019).

[19] Lingjiao Chen, Paraschos Koutris, and Arun Kumar. "Towards Model-based Pricing for Machine Learning in a Data Marketplace". In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 2019, pp. 1535–1552.

[20] Lingjiao Chen, Paraschos Koutris, and Arun Kumar. "Towards model-based pricing for machine learning in a data marketplace". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 1535–1552.

[21] Yiling Chen et al. "Optimal Data Acquisition for Statistical Estimation". In: *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*. ACM, 2018, pp. 27–44.

[22] Nadiia Chepurko et al. "ARDA: Automatic Relational Data Augmentation for Machine Learning". In: *Proc. VLDB Endow.* 13.9 (2020), pp. 1373–1387.

[23] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*. 1971, pp. 151–158.

[24] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 886–893.

[25] *Dawex*. URL: `https://www.dawex.com/en/`.

[26] *DBLP*. URL: `http://dblp.uni-trier.de/`.

[27] Vin De Silva and Joshua B Tenenbaum. *Sparse multidimensional scaling using landmark points*. Tech. rep. Technical report, Stanford University, 2004.

[28] Dong Deng, Yufei Tao, and Guoliang Li. "Overlap set similarity joins with theoretical guarantees". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 905–920.

[29] Dong Deng et al. "An efficient partition based method for exact set similarity joins". In: *Proceedings of the VLDB Endowment* 9.4 (2015), pp. 360–371.

[30] Dong Deng et al. "LCJoin: set containment join via list crosscutting". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE. 2019, pp. 362–373.

[31] Behrouz Derakhshan et al. "Optimizing Machine Learning Workloads in Collaborative Environments". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 1701–1716.

[32] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[33] Persi Diaconis and Donald Ylvisaker. "Conjugate priors for exponential families". In: *The Annals of statistics* (1979), pp. 269–281.

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

[35] Xin Luna Dong and Theodoros Rekatsinas. "Data integration and machine learning: A natural synergy". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 1645–1650.

[36] Yihe Dong et al. "Learning Space Partitions for Nearest Neighbor Search". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rkenmREFDr.

[37] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[38] Anshuman Dutt et al. "Efficiently Approximating Selectivity Functions using Low Overhead Regression Models". In: *Proc. VLDB Endow.* 13.11 (2020), pp. 2215–2228.

[39] Anshuman Dutt et al. "Efficiently approximating selectivity functions using low overhead regression models". In: *Proceedings of the VLDB Endowment* 13.12 (2020), pp. 2215–2228.

[40] Sayna Ebrahimi et al. "Uncertainty-guided Continual Learning with Bayesian Neural Networks". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[41] Jingfan Fan et al. "Adversarial similarity network for evaluating image alignment in deep learning based registration". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2018, pp. 739–746.

[42] Arash Fard et al. "Vertica-ML: Distributed Machine Learning in Vertica Database". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 755–768.

[43] Raul Castro Fernandez, Pranav Subramaniam, and Michael J Franklin. "Data market platforms: Trading data assets to solve data problems". In: *Proceedings of the VLDB Endowment* 13.12 (2020), pp. 1933–1947.

[44] Raul Castro Fernandez et al. "Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE. 2019, pp. 1190–1201.

[45] Paolo Ferragina and Giorgio Vinciguerra. "The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds". In: *Proc. VLDB Endow.* 13.8 (2020), pp. 1162–1175. URL: `http://www.vldb.org/pvldb/vol13/p1162-ferragina.pdf`.

[46] Ronald Aylmer Fisher. "Statistical methods for research workers". In: *Breakthroughs in statistics*. Springer, 1992, pp. 66–70.

[47] David Forsyth and Jean Ponce. *Computer vision: A modern approach*. Prentice hall, 2011.

[48] Rina Foygel Barber et al. "The limits of distribution-free conditional predictive inference". In: *Information and Inference: A Journal of the IMA* 10.2 (2021), pp. 455–482.

[49] Alex Galakatos et al. "Fiting-tree: A data-aware index structure". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 1189–1206.

[50] Alex Galakatos et al. "FITing-Tree: A Data-aware Index Structure". In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. Ed. by Peter A. Boncz et al., pp. 1189–1206.

[51] Chang Ge et al. "Speculative distributed CSV data parsing for big data analytics". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 883–899.

[52] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[53] Lars Gottesbüren et al. “Advanced Flow-Based Multilevel Hypergraph Partitioning”. In: *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*. Ed. by Simone Faro and Domenico Cantone. Vol. 160. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 11:1–11:15. DOI: 10.4230/LIPIcs.SEA.2020.11. URL: https://doi.org/10.4230/LIPIcs.SEA.2020.11.

[54] Yuhong Guo and Dale Schuurmans. “Discriminative batch mode active learning”. In: *Advances in neural information processing systems* 20 (2007), pp. 593–600.

[55] Maya R Gupta, Samy Bengio, and Jason Weston. “Training highly multiclass classifiers”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1461–1492.

[56] Antonin Guttman. “R-trees: A dynamic index structure for spatial searching”. In: *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 1984, pp. 47–57.

[57] Marios Hadjieleftheriou et al. “Hashed samples: selectivity estimators for set similarity selection queries”. In: *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 201–212.

[58] Jiawei Han, Micheline Kamber, and Jian Pei. “Data mining concepts and techniques third edition”. In: *The Morgan Kaufmann Series in Data Management Systems* 5.4 (2011), pp. 83–124.

[59] Xin Han, Yasushi Kawase, and Kazuhisa Makino. “Randomized algorithms for online knapsack problems”. In: *Theoretical Computer Science* 562 (2015), pp. 395–405.

[60] Hazar Harmouch and Felix Naumann. “Cardinality estimation: An experimental survey”. In: *Proceedings of the VLDB Endowment* 11.4 (2017), pp. 499–512.

[61] John A Hartigan and Manchek A Wong. “Algorithm AS 136: A k-means clustering algorithm”. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 100–108.

[62] Shohedul Hasan et al. “Deep learning models for selectivity estimation of multi-attribute queries”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 1035–1050.

[63] Alireza Heidari et al. "HoloDetect: Few-Shot Learning for Error Detection". In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 2019, pp. 829–846.

[64] Benjamin Hilprecht, Carsten Binnig, and Uwe Röhm. "Learning a Partitioning Advisor for Cloud Databases". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 143–157.

[65] Benjamin Hilprecht et al. "DeepDB: Learn from Data, not from Queries!" In: *Proc. VLDB Endow.* 13.7 (2020), pp. 992–1005. DOI: 10.14778/3384345.3384349. URL: http://www.vldb.org/pvldb/vol13/p992-hilprecht.pdf.

[66] *Hive*. URL: https://thehive.ai/.

[67] A Michael Huberman and Matthew B Miles. "Data management and analysis methods." In: (1994).

[68] David B Huberman, Brian J Reich, and Howard D Bondell. "Nonparametric conditional density estimation in a deep learning framework for short-term forecasting". In: *Environmental and Ecological Statistics* (2021), pp. 1–15.

[69] Elena Ikonomovska, Sina Jafarpour, and Ali Dasdan. "Real-time bid prediction using thompson sampling-based expert selection". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015, pp. 1869–1878.

[70] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, pp. 604–613.

[71] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[72] Matthias Jasny et al. "DB4ML - An In-Memory Database Kernel with Machine Learning Support". In: *Proceedings of the 2020 International Conference on Management*

*of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 159–173.

[73] Norman L Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous univariate distributions*. John Wiley & Sons, Ltd, 1995.

[74] Ian T Jolliffe and Jorge Cadima. "Principal component analysis: a review and recent developments". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202.

[75] Michael I Jordan and Tom M Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015), pp. 255–260.

[76] PubMed Central Journal. "https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/". In: .

[77] Aarati Kakaraparthy et al. "Optimizing Databases by Learning Hidden Parameters of Solid State Drives". In: *Proc. VLDB Endow.* 13.4 (2019), pp. 519–532. URL: http://www.vldb.org/pvldb/vol13/p519-kakaraparthy.pdf.

[78] Johan Kok Zhi Kang et al. "Efficient Deep Learning Pipelines for Accurate Cost Estimations Over Large Scale Query Workload". In: *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Ed. by Guoliang Li et al. ACM, 2021, pp. 1014–1022. DOI: 10.1145/3448016.3457546. URL: https://doi.org/10.1145/3448016.3457546.

[79] George Karypis et al. "Multilevel hypergraph partitioning: applications in VLSI domain". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7.1 (1999), pp. 69–79.

[80] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Vol. 344. John Wiley & Sons, 2009.

[81] Manohar Kaul, Bin Yang, and Christian S Jensen. "Building accurate 3d spatial networks to enable next generation intelligent transportation systems". In: *2013 IEEE 14th International Conference on Mobile Data Management*. Vol. 1. IEEE. 2013, pp. 137–146.

[82] Iman Khosravi and Seyed Kazem Alavipanah. "A random forest-based framework for crop mapping using temporal, spectral, textural and polarimetric observations". In: *International Journal of Remote Sensing* 40.18 (2019), pp. 7221–7251.

[83] Jongik Kim and Hongrae Lee. "Efficient exact similarity searches using multiple token orderings". In: *2012 IEEE 28th International Conference on Data Engineering*. IEEE. 2012, pp. 822–833.

[84] Sungyeon Kim et al. "Deep metric learning beyond binary supervision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2288–2297.

[85] Andreas Kipf et al. "Learned cardinalities: Estimating correlated joins with deep learning". In: *arXiv preprint arXiv:1809.00677* (2018).

[86] Danijel Kivaranovic, Kory D Johnson, and Hannes Leeb. "Adaptive, distribution-free prediction intervals for deep networks". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 4346–4356.

[87] Danijel Kivaranovic, Kory D. Johnson, and Hannes Leeb. "Adaptive, Distribution-Free Prediction Intervals for Deep Networks". In: *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 4346–4356. URL: `http://proceedings.mlr.press/v108/kivaranovic20a.html`.

[88] Yuqing Kong et al. "Information Elicitation Mechanisms for Statistical Estimation". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 2095–2102.

[89] *KOSARAK*. URL: `http://fimi.uantwerpen.be/data/`.

[90] Tim Kraska et al. "The case for learned index structures". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 489–504.

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

[92] Ani Kristo et al. "The Case for a Learned Sorting Algorithm". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 1001–1016.

[93] Ani Kristo et al. "The Case for a Learned Sorting Algorithm". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 1001–1016.

[94] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[95] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 2012, pp. 1106–1114.

[96] Arun Kumar et al. "To join or not to join? thinking twice about joins before feature selection". In: *Proceedings of the 2016 International Conference on Management of Data*. 2016, pp. 19–34.

[97] Jing Lei et al. "Distribution-free predictive inference for regression". In: *Journal of the American Statistical Association* 113.523 (2018), pp. 1094–1111.

[98] Daniel Lemire et al. "Roaring bitmaps: Implementation of an optimized software library". In: *Software: Practice and Experience* 48.4 (2018), pp. 867–895.

[99] Pengfei Li et al. "LISA: A Learned Index Structure for Spatial Data". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 2119–2133.

[100] Rui Li, Howard D Bondell, and Brian J Reich. "Deep distribution regression". In: *arXiv preprint arXiv:1903.06023* (2019).

[101] Yifan Li, Xiaohui Yu, and Nick Koudas. "Top-k queries over digital traces". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 954–971.

[102] Elizabeth D Liddy. "Natural language processing". In: (2001).

[103] Bing-Rong Lin and Daniel Kifer. "On arbitrage-free pricing for general data queries". In: *Proceedings of the VLDB Endowment* 7.9 (2014), pp. 757–768.

[104] Qiyu Liu et al. "Stable Learned Bloom Filters for Data Streams". In: *Proc. VLDB Endow.* 13.11 (2020), pp. 2355–2367.

[105] R. Lomasky et al. "Active Class Selection". In: *Machine Learning: ECML 2007*. 2007.

[106] David Lopez-Paz and Maxime Oquab. "Revisiting Classifier Two-Sample Tests". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[107] George S Lueker. "Average-case analysis of off-line and on-line knapsack problems". In: *Journal of Algorithms* 29.2 (1998), pp. 277–305.

[108] Lin Ma et al. "Active Learning for ML Enhanced Database Systems". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 175–191. ISBN: 9781450367356.

[109] Lin Ma et al. "Active learning for ML enhanced database systems". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 175–191.

[110] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[111] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. "An empirical evaluation of set similarity join techniques". In: *Proceedings of the VLDB Endowment* 9.9 (2016), pp. 636–647.

[112] Ryan Marcus and Olga Papaemmanouil. "Deep Reinforcement Learning for Join Order Enumeration". In: *First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. aiDM '18. Houston, TX, June 2018.

[113] Ryan Marcus and Olga Papaemmanouil. "Deep reinforcement learning for join order enumeration". In: *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 2018, pp. 1–4.

[114] Ryan Marcus, Emily Zhang, and Tim Kraska. "Cdfshop: Exploring and optimizing learned index structures". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 2789–2792.

[115] Ryan Marcus et al. "Bao: Learning to steer query optimizers". In: *arXiv preprint arXiv:2004.03814* (2020).

[116] Ryan Marcus et al. "Bao: Making Learned Query Optimization Practical". In: *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Ed. by Guoliang Li et al. ACM, 2021, pp. 1275–1288. DOI: 10.1145/3448016.3452838. URL: https://doi.org/10.1145/3448016.3452838.

[117] Ryan C. Marcus et al. "Neo: A Learned Query Optimizer". In: *Proc. VLDB Endow.* 12.11 (2019), pp. 1705–1718. DOI: 10.14778/3342263.3342644. URL: http://www.vldb.org/pvldb/vol12/p1705-marcus.pdf.

[118] Sameer Mehta et al. "How to Sell a Dataset?: Pricing Policies for Data Monetization". In: *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*. ACM, 2019, p. 679.

[119] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[120] Tova Milo and Amit Somech. "Automating Exploratory Data Analysis via Machine Learning: An Overview". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 2617–2622.

[121] Alan Mislove et al. “Measurement and Analysis of Online Social Networks”. In: *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*. San Diego, CA, Oct. 2007.

[122] Jonas Mueller and Aditya Thyagarajan. “Siamese recurrent architectures for learning sentence similarity”. In: *thirtieth AAAI conference on artificial intelligence*. 2016.

[123] Magnus Müller, Guido Moerkotte, and Oliver Kolb. “Improved selectivity estimation by combining knowledge from sampling and synopses”. In: *Proceedings of the VLDB Endowment* 11.9 (2018), pp. 1016–1028.

[124] Vikram Nathan et al. “Learning Multi-dimensional Indexes”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 985–1000.

[125] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. “Learning text similarity with siamese recurrent networks”. In: *Proceedings of the 1st Workshop on Representation Learning for NLP*. 2016, pp. 148–157.

[126] Parimarjan Negi et al. “Flow-Loss: learning cardinality estimates that matter”. In: *arXiv preprint arXiv:2101.04964* (2021).

[127] Arild Nøkland and Lars Hiller Eidnes. “Training Neural Networks with Local Error Signals”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 4839–4850.

[128] Krishna Kumar P., Paul Langton, and Wolfgang Gatterbauer. “Factorized Graph Representations for Semi-Supervised Learning from Sparse Data”. In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 1383–1398.

[129] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. “Quicksel: Quick selectivity learning with mixture models”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 1017–1033.

[130] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. "A picture of search". In: *Proceedings of the 1st international conference on Scalable information systems*. 2006, 1–es.

[131] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[132] Jian Pei. "Data Pricing–From Economics to Data Science". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 3553–3554.

[133] Jose Picado et al. "Learning Over Dirty Data Without Cleaning". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 1301–1316.

[134] Viswanath Poosala et al. "Improved histograms for selectivity estimation of range predicates". In: *ACM Sigmod Record* 25.2 (1996), pp. 294–305.

[135] Jianzhong Qi et al. "Effectively Learning Spatial Indices". In: *Proc. VLDB Endow.* 13.11 (2020), pp. 2341–2354. URL: http://www.vldb.org/pvldb/vol13/p2341-qi.pdf.

[136] Lior Rokach and Oded Maimon. "Clustering methods". In: *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 321–352.

[137] Yaniv Romano, Evan Patterson, and Emmanuel Candes. "Conformalized quantile regression". In: *Advances in neural information processing systems* 32 (2019).

[138] Yaniv Romano, Evan Patterson, and Emmanuel J. Candès. "Conformalized Quantile Regression". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 3538–3548. URL: https://proceedings.neurips.cc/paper/2019/hash/5103c3584b063c431bd1268e9b5e76fb-Abstract.html.

[139] Daniel Russo et al. "A tutorial on thompson sampling". In: *arXiv preprint arXiv:1707.02038* (2017).

[140] Alexandre Sablayrolles et al. "Spreading vectors for similarity search". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=SkGuG2R5tm.

[141] Mehran Sahami and Timothy D Heilman. "A web-based kernel function for measuring the similarity of short text snippets". In: *Proceedings of the 15th international conference on World Wide Web*. 2006, pp. 377–386.

[142] Babak Salimi et al. "Causal Relational Learning". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 241–256.

[143] David Salomon. *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[144] David Salomon and Giovanni Motta. *Handbook of data compression*. Springer Science & Business Media, 2010.

[145] Vraj Shah, Arun Kumar, and Xiaojin Zhu. "Are Key-Foreign Key Joins Safe to Avoid when Learning High-Capacity Classifiers?" In: *Proc. VLDB Endow.* 11.3 (2017), pp. 366–379.

[146] Vraj Shah, Arun Kumar, and Xiaojin Zhu. "Are key-foreign key joins safe to avoid when learning high-capacity classifiers?" In: *arXiv preprint arXiv:1704.00485* (2017).

[147] Changjian Shui et al. "Deep Active Learning: Unified and Principled Method for Query and Training". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. 2020, pp. 1308–1318.

[148] Tarique Siddiqui et al. "Cost Models for Big Data Query Processing: Learning, Retrofitting, and Our Findings". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 99–113.

[149] Abraham Silberschatz, Henry F Korth, Shashank Sudarshan, et al. *Database system concepts*. Vol. 4. McGraw-Hill New York, 1997.

[150] Panagiotis Sioulas and Anastasia Ailamaki. "Scalable Multi-Query Execution using Reinforcement Learning". In: *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Ed. by Guoliang Li et al. ACM, 2021, pp. 1651–1663. DOI: 10.1145/3448016.3452799. URL: https://doi.org/10.1145/3448016.3452799.

[151] Lukas Steinberger and Hannes Leeb. "Leave-one-out prediction intervals in linear regression models with many variables". In: *arXiv preprint arXiv:1602.05801* (2016).

[152] Ji Sun and Guoliang Li. "An End-to-End Learning-based Cost Estimator". In: *Proc. VLDB Endow.* 13.3 (2019), pp. 307–319. DOI: 10.14778/3368289.3368296. URL: http://www.vldb.org/pvldb/vol13/p307-sun.pdf.

[153] Ji Sun and Guoliang Li. "An End-to-End Learning-based Cost Estimator". In: *Proc. VLDB Endow.* 13.3 (2019), pp. 307–319. DOI: 10.14778/3368289.3368296. URL: http://www.vldb.org/pvldb/vol13/p307-sun.pdf.

[154] Ji Sun et al. "Learned Cardinality Estimation: A Design Space Exploration and A Comparative Evaluation". In: *Proc. VLDB Endow.* 15.1 (2021), pp. 85–97. URL: http://www.vldb.org/pvldb/vol15/p85-li.pdf.

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

[156] Tibor Szkaliczki. "clustering. sc. dp: Optimal clustering with sequential constraint by using dynamic programming". In: *R JOURNAL* 8.1 (2016), pp. 318–327.

[157] Natasa Tagasovska and David Lopez-Paz. "Single-Model Uncertainties for Deep Learning". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 6414–6425. URL: https://proceedings.neurips.cc/paper/2019/hash/73c03186765e199c116224b68adc5fa0-Abstract.html.

[158] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6105–6114.

[159] Matus Telgarsky and Andrea Vattani. "Hartigan's method: k-means clustering without voronoi". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 820–827.

[160] William R Thompson. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3/4 (1933), pp. 285–294.

[161] Yongxin Tong et al. "Dynamic pricing in spatial crowdsourcing: A matching-based approach". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 773–788.

[162] Immanuel Trummer. "Exact cardinality query optimization with bounded execution cost". In: *proceedings of the 2019 international conference on management of data*. 2019, pp. 2–17.

[163] Immanuel Trummer et al. "Skinnerdb: Regret-bounded query evaluation via reinforcement learning". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 1153–1170.

[164] Immanuel Trummer et al. "Skinnerdb: Regret-bounded query evaluation via reinforcement learning". In: *ACM Transactions on Database Systems (TODS)* 46.3 (2021), pp. 1–45.

[165] *Twitter's enterprise API platform*. URL: `https://developer.twitter.com/en/products/twitter-api/enterprise`.

[166] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. "Pointer networks". In: *Advances in neural information processing systems*. 2015, pp. 2692–2700.

[167] Brett Walenz et al. “Learning to Sample: Counting with Complex Queries”. In: *Proc. VLDB Endow.* 13.3 (2019), pp. 390–402. DOI: 10.14778/3368289.3368302. URL: http://www.vldb.org/pvldb/vol13/p390-walenz.pdf.

[168] Jiang Wang et al. “Learning fine-grained image similarity with deep ranking”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1386–1393.

[169] Jiannan Wang, Guoliang Li, and Jianhua Feng. “Can we beat the prefix filtering? An adaptive framework for similarity join and search”. In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 2012, pp. 85–96.

[170] Pei Wang and Yeye He. “Uni-Detect: A Unified Approach to Automated Error Detection in Tables”. In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 811–828.

[171] Xiaoying Wang et al. “Are We Ready For Learned Cardinality Estimation?” In: *Proc. VLDB Endow.* 14.9 (2021).

[172] Xiaoying Wang et al. “Are We Ready For Learned Cardinality Estimation?” In: *Proc. VLDB Endow.* 14.9 (2021), pp. 1640–1654. URL: http://www.vldb.org/pvldb/vol14/p1640-wang.pdf.

[173] Xubo Wang et al. “Leveraging set relations in exact and dynamic set similarity join”. In: *The VLDB Journal* 28.2 (2019), pp. 267–292.

[174] Zheng Wang et al. “Efficient and Effective Similar Subtrajectory Search with Deep Reinforcement Learning”. In: *Proc. VLDB Endow.* 13.11 (2020), pp. 2312–2325.

[175] *WorldQuant*. URL: https://data.worldquant.com.

[176] Peizhi Wu and Gao Cong. “A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation”. In: *SIGMOD ’21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Ed. by Guoliang Li et al. ACM, 2021, pp. 2009–2022. DOI: 10.1145/3448016.3452830. URL: https://doi.org/10.1145/3448016.3452830.

[177] Peizhi Wu and Gao Cong. "A unified deep model of learning from both data and queries for cardinality estimation". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2009–2022.

[178] Renzhi Wu et al. "Learning to be a Statistician: Learned Estimator for Number of Distinct Values". In: *Proc. VLDB Endow.* 15.2 (2021), pp. 272–284. URL: http://www.vldb.org/pvldb/vol15/p272-wu.pdf.

[179] Wentao Wu, Jeffrey F Naughton, and Harneet Singh. "Sampling-based query re-optimization". In: *Proceedings of the 2016 International Conference on Management of Data*. 2016, pp. 1721–1736.

[180] Wentao Wu et al. "Predicting query execution time: Are optimizer cost models really unusable?" In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE. 2013, pp. 1081–1092.

[181] Chuan Xiao et al. "Top-k set similarity joins". In: *2009 IEEE 25th International Conference on Data Engineering*. IEEE. 2009, pp. 916–927.

[182] Chuan Xiao et al. "Efficient similarity joins for near-duplicate detection". In: *ACM Transactions on Database Systems (TODS)* 36.3 (2011), pp. 1–41.

[183] *Xignite*. URL: https://aws.amazon.com/solutionspace/financial-services/solutions/xignite-market-data-cloud-platform/.

[184] Xi Yan, David Acuna, and Sanja Fidler. "Neural Data Server: A Large-Scale Search Engine for Transfer Learning Data". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 3892–3901.

[185] Jaewon Yang and Jure Leskovec. "Defining and evaluating network communities based on ground-truth". In: *Knowledge and Information Systems* 42.1 (2015), pp. 181–213.

[186] Lei Yang et al. "Leaper: A Learned Prefetcher for Cache Invalidation in LSM-tree based Storage Engines". In: *Proc. VLDB Endow.* 13.11 (2020), pp. 1976–1989.

[187] Zongheng Yang et al. "Deep Unsupervised Cardinality Estimation". In: *Proc. VLDB Endow.* 13.3 (2019), pp. 279–292. DOI: 10.14778/3368289.3368294. URL: http://www.vldb.org/pvldb/vol13/p279-yang.pdf.

[188] Zongheng Yang et al. "NeuroCard: One Cardinality Estimator for All Tables". In: *CoRR* abs/2006.08109 (2020).

[189] Zongheng Yang et al. "Qd-tree: Learning data layouts for big data analytics". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 193–208.

[190] Xiang Yu et al. "Reinforcement learning with tree-lstm for join order selection". In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE. 2020, pp. 1297–1308.

[191] Ji Zhang et al. "An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning". In: *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 2019, pp. 415–432.

[192] Meng Zhang et al. *Optimal and Quantized Mechanism Design for Fresh Data Acquisition*. 2020. arXiv: 2006.15751.

[193] Yong Zhang et al. "An efficient framework for exact set similarity search using tree structure indexes". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE. 2017, pp. 759–770.

[194] Yong Zhang et al. "A Transformation-Based Framework for KNN Set Similarity Search". In: *IEEE Trans. Knowl. Data Eng.* 32.3 (2020), pp. 409–423.