Accelerating Human-in-the-loop Data Analytics

by

Jinglin Peng

B.Sc., Harbin Institute of Technology, 2016

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

Declaration of Committee

Name:	**Jinglin Peng**
Degree:	**Doctor of Philosophy**
Thesis title:	**Accelerating Human-in-the-loop Data Analytics**
Committee:	**Chair:** Yuepeng Wang Assistant Professor, Computing Science

Jiannan Wang
Supervisor
Associate Professor, Computing Science

Jian Pei
Committee Member
Professor, Computing Science

Tianzheng Wang
Examiner
Assistant Professor, Computing Science

Ke Yi
External Examiner
Professor
Department of Computer Science and Engineering
Hong Kong University of Science and Technology

Abstract

Data analytics is essential to enable data-driven decision-making. While batch analytics is often run offline and can take several hours or even days to generate results, human-in-the-loop analytics requires a fast response. However, it is still a challenging problem to accelerate human-in-the-loop data analytics. The challenge comes from both machine and human sides. From the machine side, there is a gap between the massive volume of processed data and limited hardware resources, which is constrained by practical considerations like price. From the human side, a gap exists between the little human attention and the enormous details that the attention needs to be paid to finish a task.

In this thesis, we develop several systems to accelerate human-in-the-loop data analytics from both machine and human sides. The thesis contains two parts. In the first part of the thesis, we present two systems (AQP++ and SamComb) to accelerate machine processing. In order to achieve interactive response time, our key idea is to reduce the data that needs to be processed by the machine. We focus on the online analytical processing (OLAP) scenario and leverage sampling-based approximate query processing (AQP) techniques to reduce the data. An AQP system can return an approximate query result in a short time. To improve the estimation quality, we propose to combine different data summaries: In AQP++, we combine samples with pre-computed aggregations; In SamComb, we combine different types of samplers. In the second part of the thesis, we present one system to accelerate human analytics. We focus on the exploratory data analysis (EDA) scenario and propose a task-centric EDA system named DataPrep.EDA. DataPrep.EDA allows data scientists to declaratively specify a wide range of EDA tasks with a single function call. In this way, humans can pay more attention to deciding the task to perform, and the system will handle the implementation details automatically.

Keywords: Data Analytics; Human-in-the-loop; Approximate Query Processing; Exploratory Data Analysis

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Jiannan Wang. Jiannan has introduced me to research and taught me skills from almost all aspects of being a good researcher: from finding a good problem to maximizing the influence of work. Jiannan cared about both my research and my personal life deeply. He was one of the nicest people I have met. He always encouraged me whenever I faced difficulty and provided me with the best support.

I would like to thank Prof. Jian Pei, Prof. Tianzheng Wang, Prof. Ke Yi and Prof. Yuepeng Wang for serving on my thesis committee. They helped examine the thesis and provided insightful feedback.

I was fortunate to meet a group of supportive mentors and collaborators. I would like to thank Dr. Bolin Ding and Dr. Kai Zeng, who kindly hosted me while I did research internships at Alibaba Group, shared many valuable and practical ideas and gave great feedback on my research. I would like to thank Prof. Jeffrey M. Rzeszotarski for collaborating on DataPrep.EDA project. He provided insightful advice on the user study design and experiment.

I was fortunate to spend time with excellent peers. I would like to thank my lab mates: Pei Wang, Weiyuan Wu, Changbo Qu, Danrui Qi, Xiaoying Wang, Chunyu Chen, Andy Zhang, Yi Xie, Ruocheng Jiang, Yongjun He, Song Bian, Xi Yang, Lydia Zheng, Brandon Lockhart and Mathew Teoh. The fantastic journey with you was an unforgettable memory in my life. A special thanks to Jing Nathan Yan for the continuous help from different aspects and the nice cooking. Special thanks to Yaliang Li and Tianhao Wang for the wonderful time spent at Bellevue.

Finally, I would like to thank my parents for their unconditional love and tolerance. I could not finish my Ph.D. study without their support and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Data analytics is essential to enable data-driven decision-making. Two types of data analytics exist: batch analytics is often run offline and can take several hours or even days to generate results, and human-in-the-loop analytics requires fast response and needs to get results in seconds or minutes. In this thesis, we focus on human-in-the-loop data analytics.

For human-in-the-loop analytics, a fast response is crucial. A study shows that a delay longer than 500ms *"incurred significant costs, decreasing user activity and data set coverage while reducing rates of observation, generalization and hypothesis."* [113]. Despite its importance, accelerating human-in-the-loop analytics remains a challenging problem. In human-in-the-loop analytics, both humans and machines are involved. The human sends a computational request to the machine, then the machine processes the data and request and sends the result to the human. After that, the human analyzes the result and comes up with the following action. During this process, the time of finishing a task can be divided into two parts: machine processing (i.e., machine processes the data and request) and human analytics (i.e., human analyzes the result and thinks about the following action). The challenge of accelerating data analytics comes from both machine and human sides.

1.1 Challenges

Machine Side: Big Data vs. Limited Resources. From the machine side, there exists a gap between the massive volume of the data and the limited hardware resources, which is also observed in [142]. Nowadays, the data is growing at an unprecedented rate. For example, Google ingests more than two terabytes of data per second in Monarch (Google's in-memory time series database) in July 2019 [45]. While the data grows fast, the usable resources are constrained by practical considerations like the budget. Besides, even if we have an unlimited budget to increase as many resources as we want, the system also faces the challenge of interactive response when dealing with a large amount of data: due to the time constraint, the latency needs to scale sub-linearly with the data size.

Human Side: Enormous Details vs. Limited Attention. From the human side, there exists a gap between the limited human attention and the enormous details that need to be handled in data analytics. An example can be found in Example 1. In this example, human needs to pay attention to 1) what goal I want to achieve (gain an overview understanding), 2) what output I want to get from the machine (a histogram for each numerical attribute, a box plot for each categorical attribute, and a count of missing values for each attribute), and 3) the implementation details of getting those plots and statistics (e.g., what tool to use, how to implement and customize the visualization). The enormous details can easily make people lose their attention and slow down the whole process of data analytics.

Example 1. *Alice collects a dataset of houses and wants to build a model to predict the house price. When she explores the data, she first wants to understand the dataset, such as how each attribute looks like and whether they contain missing values. To understand the distributions, Alice plots a histogram for each numerical attribute and a bar chart for each categorical attribute. Meanwhile, she writes code to count the missing values of each attribute. After knowing the basic information, Alice finds that some attributes are not helpful for price prediction, including an ID attribute, an attribute that contains only a single category, and two attributes that contain too many missing values. She then drops those attributes and continues her analysis.*

1.2 Overview of Contributions

In this thesis, we develop systems to accelerate data analytics from both machine and human sides. From the machine side, our key idea is to reduce the data that needs to be processed by the machine. More specifically, we focus on the online analytical processing (OLAP) scenario and leverage sampling-based approximate query processing (AQP) techniques to approximately process analytical queries (e.g., SUM, COUNT, AVG, ...). In an AQP system, users can make a trade-off between the answer quality and response time by tuning the sample size. To further improve the estimation quality, we developed two systems with a key idea of combining multiple data summaries: 1) in AQP++, we combine samples with cubes that store pre-computed aggregations, and 2) in SamComb, we combine different types of samplers. From the human side, our key idea is to accelerate human analytics through declarative APIs. We focus on the exploratory data analysis (EDA) scenario and propose a task-centric EDA system named DataPrep.EDA. In DataPrep.EDA, each API is mapped to an EDA task. In this way, humans can pay more attention to identifying the task to perform and leave the implementation details to the system.

Our systems are mainly designed for data scientists who have some basic knowledge of statistics and programming: for the two AQP systems, the users should know the meaning of the confidence interval, which is used to measure the quality of the returned estimation; for the DataPrep.EDA system, the users should be able to conduct exploratory data analysis by writing Python code in Jupyter notebook.

1.2.1 Accelerating Machine Processing by Approximate Computation

In the first part of this thesis, we present two systems that accelerate machine processing by answering queries approximately.

There exist two approaches to speed up machine processing. The first approach is to build a fast engine and make better use of hardware. Many systems (e.g., Spark [175], Presto [147] and Flink [59]) have been developed in this direction and are commonly used for big data analytics. This approach returns the exact result (rather than the approximate result). However, it is hard to make the latency scale sub-linearly with the data size, which makes it hard to achieve interactive responses for large-scale data. The second approach is to only process a summary of the data (e.g., a sample) and return an approximate result. Although the result is approximate, this approach can control the amount of processed data, making it easier to achieve interactive responses. Due to the importance of interactive responses in human-in-the-loop data analytics, we focus on this direction.

As discussed before, we leverage sampling-based approximate query processing (AQP) to process analytical queries. It estimates the query result using a sample of the data rather than the data itself. Users can make a trade-off between the response time and answer quality by tuning the sample size. In order to achieve interactive response time, it is a common choice to use a small sample. However, a small sample usually gives a bad estimation. It remains a challenging problem to improve the estimation quality while keeping interactive response time. To alleviate this problem, we observed that different data summaries that are designed for different types of queries could usually achieve a better trade-off between response time and answer quality for their suitable queries. Therefore, we propose a `Sample + X` framework, which combines samples with other data summaries. The sample is a general data summary from which most queries can benefit, and **X** is a specialized data summary from which its suitable queries can benefit. We then developed two systems based on this idea: AQP++ [135] combines samples and precomputed aggregations, and SamComb [133] combines different types of samplers.

AQP++: Combining Samples and Precomputed Aggregations. Apart from sampling, aggregate precomputation (AggPre) is another commonly used approach to avoid scanning the full data. It precomputes the answers of some aggregation queries and stores the results in a structure such as the prefix-sum cube [91]. Although AggPre can achieve interactive response time and give the exact result, it has a high preprocessing cost, which increases exponentially with the number of dimensions. On the other hand, sampling does not suffer from the curse of dimensionality, but it gives an approximate result. Then, the question is can we combine these two approaches to make a flexible trade-off among preprocessing cost, estimation quality and response time? To enable the combination of samples and AggPre, we propose the AQP++ framework. The framework can leverage both a sample as well as a precomputed aggregate to answer user queries. We discuss the advantages of having such a unified framework and identify new challenges to fulfill this vision. We also conduct an in-depth study of these challenges for range queries and explore both optimal and heuristic solutions to address them. Our experiments using two public benchmarks and one real-world dataset show that AQP++ achieves a more flexible and better trade-off among preprocessing cost, query response time, and answer quality than AQP or AggPre.

SamComb: Combining Different Types of Samplers. AQP++ leverages cubes to improve the answer quality of sample-based estimation. However, the cube only supports range queries and suffers from the curse of dimensionality. On the other hand, the sample is a more general data summary and does not suffer from the curse of dimensionality. Therefore, we also study whether we can combine different types of samples. Various samplers have been proposed (e.g., uniform sampler, stratified sampler, and measure-biased sampler) since no single sampler works well in all cases. This also motivates us to combine different samplers to solve the "one size does not fit all" issue. We then propose SamComb, a novel bandit-based sampler combination framework. Given a set of samplers, a budget, and a query, SamComb can automatically decide how much budget should be allocated to each sampler so that the combined estimation achieves the highest accuracy. We model this sampler combination problem as a multi-armed bandit (MAB) problem and propose effective approaches to balance the exploration and exploitation trade-off in a principled way. We provide theoretical guarantees for our approaches and conduct extensive experiments on synthetic and real datasets. The results show a strong need to combine multiple samplers to obtain accurate estimations without knowledge about population predicates and distributions, and SamComb is an effective framework to achieve this goal.

1.2.2 Accelerating Human Analytics by Task-Centric API Design

In the second part of this thesis, we present DataPrep.EDA [134], a system to accelerate human analytics by task-centric API design.

DataPrep.EDA: A task-centric EDA System in Python. We focus on the exploratory data analysis (EDA) scenario, which is a crucial step in any data science project. Currently, existing Python libraries fall short of supporting data scientists to complete common EDA tasks for statistical modelling. Their API design is either too low level, optimized for plotting rather than EDA, or too high level, which is hard to specify more fine-grained EDA tasks. In response, we propose DataPrep.EDA, a novel task-centric EDA system in Python. DataPrep.EDA allows data scientists to declaratively specify a wide range of EDA tasks in different granularity with a single function call. We identify several challenges to implementing DataPrep.EDA and propose effective solutions to improve the system's scalability, usability, and customizability. In particular, we discuss some lessons learned from using Dask to build the data processing pipelines for EDA tasks and describe our approaches to accelerate the pipelines. We conduct extensive experiments to compare DataPrep.EDA with Pandas-profiling [56], the state-of-the-art EDA system in Python. The experiments show that Pandas-profiling significantly outperforms Pandas-profiling in terms of speed and user experience.

1.2.3 Thesis Organization

The rest of the thesis is organized as follows. We introduce the background context and related work in Chapter 2. In Chapter 3 and Chapter 4, we present AQP++ and SamComb, which accelerate machine processing by approximate query processing. Then we introduce DataPrep.EDA in Chapter 5. It accelerates human analytics in exploratory data analytics scenarios by task-centric API design. Finally, we conclude the thesis and discuss the future work in Chapter 6.

Chapter 2

Background and Related Work

In this chapter, we provide background context and briefly review related works.

2.1 Related Thesis

The most related thesis of this work is [142], which also studied how to improve machine and human efficiency. This work and [142] focused on different scenarios and solved the problem with different ideas. For improving machine efficiency, [142] focused on the big data scenario (where data is stored in partitions) and KDE scenario, with a key idea of combining precomputation and query-time sampling. Different from it, we focused on the common OLAP scenario with the key idea of combining multiple data summaries. We did not assume the way of doing sampling (e.g., samples are pre-created in AQP++). For improving human efficiency, [142] focused on the monitoring scenario and studied how to prioritize human attention to the important subset of data. Different from it, we focused on the exploratory data analysis scenario and saved human attention by leaving the implementation details to systems.

2.2 Approximate Query Processing

Sampling-based AQP has been extensively studied in the last several decades [122, 66, 73]. There exist two types of AQP systems based on when samples are created: 1) samples are pre-created, and 2) samples are created on the fly. In this section, we focus on AQP systems that pre-create samples.

In an AQP system, users can make a trade-off between estimation quality and response time by tuning the sample size. When a query comes, the AQP system estimates the query result and calculates the corresponding confidence interval under a customizable confidence level, which can measure the estimation quality. For example, suppose the confidence level is 95%. If the AQP system returns 300 ± 50, then it means that the real result lies in the interval $[300-50, 300+50]$ with a probability of 95%.

Uniform Sampling. The simplest sampling approach is uniform sampling. It samples each tuple with an equal probability. Next, let us use a uniform sample to illustrate how the AQP system works. The AQP system usually supports queries in the following form:

```
SELECT f(A) FROM table
WHERE Predicate (B1, B2, ...)
GROUP BY C1, C2, ...
```

where f is an aggregation function, such as `SUM`, `COUNT` and `AVG`, A is a numerical attribute, and `Predicate` is a query predicate.

Suppose a user issues the following query:

```
SELECT SUM(price) FROM D
```

Let D be the population and S be the uniform sample. Then the sampling ratio is $|S|/|D|$, where $|S|$ is the sample size and $|D|$ is the table size. The AQP system estimates the query answer as:

$$est \texttt{ = SELECT } |D| \cdot \frac{\texttt{SUM(price)}}{|S|} \texttt{ FROM } S\texttt{.}$$

For example, if the sampling ratio is 1%, `SUM(price)` will be multiplied by 100. The estimation will be more accurate with a larger sample size. The confidence interval of the estimation (i.e., est) can be computed using either Central Limit Theorem (CLT) or Bootstrapping. The former is more efficient to calculate, while the latter can support a broader range of aggregation functions, including user-defined functions. The confidence interval of a SUM query can be derived from CLT:

$$\epsilon \texttt{ = SELECT } \lambda \cdot |D|\sqrt{\frac{\texttt{VAR(price)}}{|S|}} \texttt{ FROM } S\texttt{,}$$

where λ is a parameter determined by a user-specified confidence level (e.g., $\lambda = 1.96$ for a 95% confidence level).

Due to the simplicity and generality of uniform samples, they are widely used by existing AQP systems [131, 47]. However, since the uniform sampling takes each tuple with an equal probability, the rare groups may be under-represented in the sample.

Stratified Sampling. Stratified sampling is a commonly used approach to improve estimation quality. It divides the population into strata (i.e., partitions) and performs a uniform sampling within each stratum. It first estimates within each stratum during query processing and then combines all estimations.

The performance of stratified sampling mainly depends on two factors: 1) how data is partitioned; and 2) how large the sample of each stratum is. The system can be optimized for a query workload by tuning the stratified sampling design. For example, STRAT [64] partitions the data based on a collection of historical queries to optimize the performance over the workload. Given a total budget when creating samples, STRAT leverages an allocation strategy that optimizes the estimation quality for the query workload. The strategy is based on the Neyman allocation [170]. I.e., more budgets are allocated to the stratum with a large variance and population size.

Biased Sampling. Uniform sampling takes each tuple with an equal probability. However, tuples do not contribute to the result equally. For example, for a sum query, a tuple with a large value contributes more to the final result. Based on this observation, biased sampling is proposed. Biased sampling samples each tuple with a different probability and assign more weights to the tuples that are important to the result.

Sample + Seek [78] has applied biased sampling to improve the performance. Given a collection of measure columns, Sample + Seek draws biased samples. Each tuple is drawn with a probability proportional to its measure value. Let t be a tuple and M be a measure column, the probability of t appears in the sample is:

$$Pr(t) = \frac{t_M}{\sum_{t' \in D} t'_M}, \tag{2.1}$$

Since the probability is proportional to the measured value, this technique is called measure-biased sampling. It is similar to the proportional-to-size sampling schema in statistics [155]. Clearly, tuples with larger values have a higher impact on the query result. Since they are sampled with a higher probability in the measure-biased sampling schema, measure-biased sampling can improve the estimation quality.

2.3 Exploratory Data Analysis

Exploratory data analysis (EDA) is an essential step in the data processing pipeline, providing necessary data profiling and insight discovery before in-depth analysis [167]. In this section, we review existing EDA tools.

EDA Tools in Python and R. Python and R are the two most popular programming languages in data science. Similar to Python, there are many EDA libraries in R, including DataExplorer [13] and visdat [159] (see [157] for a recent survey). However, they are either similar to Pandas+Plotting or Pandas-profiling, thus having the same limitations as them. In the database community, recently, there has been a growing interest in building EDA systems for Python programmers to benefit a large number of real-world data scientists [76, 23]. To the best of our knowledge, DataPrep.EDA is the first task-centric EDA system in Python and the only EDA system explicitly dedicated to the notion of task-centric EDA.

GUI-based EDA. A GUI-based environment is commonly used for doing EDA, particularly among non-programmers. In such an environment, an EDA task is triggered by a click, drag, drop, etc. (rather than a Python function call). Many commercial systems including Tableau [35], Excel [24], Spotfire [38], Qlik [28], Splunk [32], Alteryx [6], SAS [30], JMP [21] and SPSS [20] support doing EDA using a GUI. Although these systems are suitable in many cases, they all have the fundamental limitations of being removed from the programming environment and lacking flexibility.

In recent years, there has been abundant research in visualization recommendation systems [168, 169, 116, 74, 152, 77, 92, 115, 121]. Visualization recommendation is the process of automatically determining an interesting visualization and presenting it to the user. Another related area is automated insight generation (auto-insights). An auto-insight system mines a dataset for statistical properties of interest [79, 112, 158, 25, 75, 160]. Unlike these systems, DataPrep.EDA is a programming-based EDA tool with several advantages over GUI-based EDA systems, including seamless integration in the Python data science ecosystem and flexibility since the data scientist is not restricted to one GUI's functionalities.

Data Profiling. Data profiling is the process of deriving summary information (e.g., data types, the number of unique values in columns) from a dataset (see [42] for a data profiling survey). Metanome [130] is a data profiling platform where users can run profiling algorithms on their data to generate different summary information. Data profiling can be used in the tasks of data quality assessment (e.g., Profiler [101]) and data cleaning (e.g., Potter's Wheel [141]). Although DataPrep.EDA performs data profiling, unlike the above systems, it is integrated effectively into a Python programming environment.

Python data profiling tools, including Pandas-profiling [56], Sweetviz [34], and AutoViz [8], enable profiling a dataset by running one line of code. These systems provide rigorous and coarse-grained analyses, which are unsuitable for general, ad-hoc EDA.

Part I

Accelerate Machine Processing by Approximate Computation

Chapter 3

AQP++: Combining Samples and Precomputed Aggregations

This chapter presents AQP++, a unified framework to combine samples and precomputed aggregations. It allows users to make a flexible trade-off among preprocessing cost, response time and answer quality.

3.1 Motivation

As discussed in Chapter 1, we aim to accelerate machine processing by avoiding scanning the full data. In the past, the database community proposed two *separate* ideas to achieve this goal. One is *sampling-based approximate query processing* (AQP) [47, 44, 64]), which creates a random sample of data and uses the sample to estimate query results. The other is *aggregate precomputation* (AggPre) such as data cubes [89, 86, 91, 124], which pre-computes the answers to some aggregation queries and then uses the pre-computed aggregates to speed up query performance. In comparison, although both approaches can answer queries fast, they achieve the goal via different trade-offs. Essentially, AQP varies sample size to balance the trade-off between answer quality and query response time; AggPre varies the size of pre-computed aggregates to balance the trade-off between *preprocessing cost* (i.e., the time and space used to calculate and store aggregates) and query response time.

AQP has been extensively studied in the past [127, 44, 43, 83, 90, 85, 62, 64, 96] and there has been a resurgence of interest in recent years [47, 46, 125, 176, 177, 108, 78, 108, 66, 164, 107, 69, 132, 103, 123]. However, to the best of our knowledge, none of existing AQP work has sought a *general* framework to connect AQP with AggPre (see Section 3.2 for a detailed comparison).

There are (at least) two strong motivations for building such a connection. Firstly, it is often the case that a data warehouse has already precomputed the answers to a large number of aggregation queries (e.g., data cubes). Suppose one wants to apply AQP to the data warehouse to reduce the query response time. Without building the connection with

AggPre, AQP will miss the opportunity to leverage the (free) precomputed aggregates to improve its query performance. Secondly, as mentioned above, AQP and AggPre implement interactive analytics through different trade-offs (i.e., answer quality vs. response time, preprocessing cost vs. response time). Connecting AQP with AggPre will allow a more flexible and better trade-off between preprocessing cost, query response time, and answer quality, and thus is more likely to satisfy user needs.

To this end, we propose AQP++, a general framework to connect AQP with AggPre for interactive analytics. The key idea of AQP++ is that, unlike AQP, which uses a sample to directly estimate a query result, it uses a sample to estimate the difference of the query result from a pre-computed aggregate. Consider a simple example. Suppose we want to estimate the answer to the following query q.

$q(\mathcal{D})$: `SELECT SUM(`A`) FROM` $\mathcal{D}$ `WHERE` $1 < C < 10$.

Suppose the answer to the following aggregate query, *pre*, has been precomputed.

$pre(\mathcal{D})$: `SELECT SUM(A) FROM` $\mathcal{D}$ `WHERE` $2 < C < 10$.

To estimate $q(\mathcal{D})$, AQP++ first uses a sample to estimate the difference of $q(\mathcal{D})$ from $pre(\mathcal{D})$, i.e.,

$q(\mathcal{D})$-$pre(\mathcal{D})$: `SELECT SUM(A) FROM` $\mathcal{D}$ `WHERE` $1 < C \leq\ 2$.

Then, it adds the estimated difference to the known $pre(\mathcal{D})$ to obtain the estimation of $q(\mathcal{D})$.

We find that AQP++ framework is very general because (1) it works for a wide range of aggregate functions such as SUM, COUNT, AVG, and VAR; (2) it can be easily extended to support many optimization techniques (that were originally proposed for AQP) such as stratified sampling [43, 62, 50, 63, 64, 153, 47, 83] and outlier indexing [62]. Furthermore, for any aggregation query, if AQP can estimate its result unbiasedly, AQP++ can return an unbiased estimate as well.

To quantify the uncertainty of an estimated answer, we investigate how AQP++ can compute a confidence interval and demonstrate analytically and empirically when AQP++ can return a tighter confidence interval (i.e., a more accurate answer) than AQP. Specifically, we treat the estimated answers to a user query and a pre-computed query as two random variables. We find that the more correlated the two random variables are, the more accurate answer AQP++ can return. In extreme cases, if a user query and a pre-computed query are identical, with a correlation coefficient of 1, AQP++ will return the exact answer.

To examine whether AQP++ can achieve a better trade-off among preprocessing cost, response time, and answer quality compared to AQP or AggPre alternatives, we conduct an in-depth study of AQP++ for *range queries*. A range query applies an aggregation operation over all records selected by a conjunction of range conditions. We choose this form of queries because (1) it is a critical class of queries for analytical workloads, and (1) it can not only be well supported by AQP but also has been extensively studied in the AggPre literature [91, 61, 84, 71, 111]. The study requires solving two challenging problems.

- *Aggregate Identification.* Given a user query, there might be a large number of precomputed aggregate values available. How should we quickly identify the best one for the query?
- *Aggregate Precomputation.* Pre-computing all possible aggregate values is prohibitively expensive. Given a space budget, how should we decide which aggregate values to be pre-computed?

We use a simple example to illustrate the challenges of the two problems as well as our contributions made to address them. Suppose a user wants to interactively issue a query in the following form:

SELECT SUM(A) FROM $\mathcal{D}$ WHERE $x \leq C \leq y$,

where $x, y = 1, 2, \cdots, 100$. An efficient AggPre approach is to precompute a *prefix cube* [91]. In this example, the (1-dimensional) prefix cube consists of 100 cells:

SELECT SUM(A) FROM $\mathcal{D}$ WHERE $C \leq t$

where $t = 1, 2, \cdots, 100$. Once the cube is created, it is easy to see that any user query can be answered by accessing at most 2 cells in the cube. Since there are $\frac{101 \cdot 100}{2} = 5050$ different user queries, the pre-computed cube (with only 100 cells) can be thought of as containing 5050 aggregate values.

Let us first consider the aggregate-identification problem. Suppose that only ten cells (e.g., $t = 10, 20, \cdots, 100$) in the cube can be stored. In this situation, the cube contains $\frac{11 \cdot 10}{2} = 55$ aggregate values. Given a query, e.g.,

SELECT SUM(A) FROM $\mathcal{D}$ WHERE $15 \leq C \leq 41$,

AQP++ needs to decide which one of the following 55 precomputed values should be used to estimate the answer to the above query.

SELECT SUM(A) FROM $\mathcal{D}$ WHERE $x \leq C \leq y$

where $x, y = 10, 20, \cdots, 100$.

Since the query's answer quality highly depends on the identified value, the decision has to be made carefully. But, we cannot afford to try out all aggregate values as it will increase query processing time significantly. To this end, we propose an efficient aggregate-identification approach. The key idea is to quickly identify a small number of aggregate values whose corresponding queries look similar to the user query and then examine which one is the best only among them. We prove the optimality of this approach under certain assumptions and show that it achieves good performance empirically in the general setting.

Next, let us turn to the aggregate-precomputation problem. Suppose the space budget is only available for creating a prefix cube with ten cells. Our goal is to decide which 10 out of 100 cells should be selected to precompute. Note that there are $\binom{100}{10}$ different ways to choose the ten cells, so a brute-force search will not work. We formally define the problem and find that an *equal-partition scheme* (i.e., $t = 10, 20, \cdots, 100$) is not always optimal. To solve the problem, we find that two factors, attribute correlation and data distribution, may affect the cell-selection decision. We first prove that the equal-partition scheme is optimal if attributes A and C are independent and C has no duplicate values. The theoretical result guides us to develop a hill-climbing-based heuristic approach that can adjust the partition scheme *adaptively* based on the actual attribute correlation and data distribution.

Please note that the above simple example demonstrates the challenges for one-dimensional range queries (i.e., with a single range condition). There are many other challenges involved when dealing with multidimensional cases. Later, we will discuss these challenges in detail and propose effective approaches.

In summary, we make the following contributions:

- We argue that the two different ideas for interactive analytics, AQP and AggPre, should be connected and propose AQP++, the first general framework to enable the connection.
- We conduct an in-depth study of AQP++ for range queries and formalize two challenging problems in the study: aggregate identification and aggregation precomputation.
- We develop an efficient aggregate-identification approach and show the approach's effectiveness analytically and empirically.
- We identify two critical factors that affect the solution to the aggregation-precomputation problem. We prove that the equal-partition scheme is only optimal under certain assumptions and propose an effective hill-climbing approach for general situations.
- We evaluate AQP++ using a commercial OLAP system on three datasets. Experimental results show that AQP++ can achieve up to 10x more accurate answers than AQP for a relatively small preprocessing cost.

The remainder of this chapter is organized as follows. We review related work in Section 3.2. In Section 3.3, we formally define the aggregate identification and aggregate precomputation problems. To address these problems, we first present the AQP++ framework in Section 3.4 and then propose effective approaches for them in Section 3.5 and Section 3.6, respectively. We describe the results of our experimental studies in Section 3.8 and present conclusions and future work in Section 3.9.

3.2 Related Work

Approximate Query Processing. Sampling-based AQP has been extensively studied in the last several decades [122, 66, 73]. A lot of techniques have been proposed to optimize AQP's performance. Most of them are focused on generating better *stratified samples* [43, 62, 50, 63, 64, 153, 47, 83]. There has also been some work that tries to augment samples with auxiliary indices [62, 78, 120]. Unlike them, AQP++ is a framework that can connect *any* existing AQP engine with AggPre. Thus, all these techniques can be easily extended to the AQP++ framework (see Section 3.4.2 for a detailed discussion).

AQP++ is similar in spirit to some recent work [132, 82], which observe that previous answers can be beneficial to estimate the answers to future queries. AQP++ is fundamentally different from these work in two aspects: (1) AQP++ utilizes precomputed exact answers over full data rather than approximate answers over samples to improve future queries; (2) AQP++ uses a sample to estimate the difference between the answers to a previous query and a future query, rather than build a model to predict unobserved data points.

Our work is also related to *Approximate Pre-Aggregation* [97, 94], which combines samples with a small set of statistics of the data to improve answer quality. However, they assume that the set of statistics is available in the system without considering how to precompute a BP-Cube and how to use it for result estimation.

Online aggregation [90, 108, 103, 176, 144, 129, 172, 143] is another popular application scenario of AQP systems, which progressively improves answer quality by dynamically increasing sample size. In this, we consider the scenario where samples are created before queries. Still, it would be an interesting future direction to investigate the use of AQP++ in an online-sampling setting.

In fact, AQP++ was initially inspired by SampleClean [164, 107]. SampleClean enables fast and accurate query processing on dirty data. Its basic idea is to clean a sample of dirty data and then use the cleaned sample to correct dirty query results. Specifically, SampleClean is given one query, and the goal is to estimate the difference between its results computed on two datasets (a dirty dataset and a cleaned dataset). In contrast, AQP++ is given two queries (a user query and a pre-computed query), and the goal is to estimate the difference between their results computed on one dataset.

In addition to sampling-based AQP, several non-sampling-based techniques have been proposed recently [139, 58]. They aim to support more complex queries and provide deterministic guarantees using indices rather than samples. However, the query class we support they are not as effective as sampling-based AQP.

Aggregate Precomputation. Aggregate precomputation is another extensively studied topic in the database community [150, 119, 124, 86, 91]. Data cubes, which store data aggregated across all possible combinations of dimensions, are widely used in data warehouses [124, 65]. Since it is often very expensive to store a complete data cube [151], there is some work on partial data cube precomputation [80, 89, 80]. There is also some work that tries to apply sampling or approximation techniques to data cubes [163, 110, 95, 51, 99, 100]. For example, Li et al. [110] studied how to compute the confidence intervals for a cube constructed from a sample. Vitter and Wang [163] proposed an I/O efficient technique to build an approximate data cube based on wavelets. Compared with AQP, these techniques still suffer from (1) much higher preprocessing cost or/and (2) not good at answering ad-hoc queries. Thus, having a unified framework like AQP++ is more desirable.

Materialized Views. Precomputed query results are also known as materialized views (see [70] for a survey). If we look at AQP++ in a materialized view context. aqpp essentially materializes aggregation views [150] as well as sample views [98], and studies how to answer queries using the materialized views. The two problems (aggregate identification and aggregate precomputation) that this work delves into are known as answering queries using views [87, 72] and selecting views to materialize [150, 89]. But, existing approaches cannot be used to solve our problems because none of them has considered the connection between AQP and AggPre.

3.3 Problem Formalization

This section formally defines our problems. For ease of presentation, we assume that our queries do not have a group-by clause. The extension to group-by queries will be discussed in Appendix 3.7.

Definition 1 (Query Template)**.** *A query template, denoted by* $Q : [f(A), C_1, C_2, \cdots C_d]$*, represents a collection of queries of the following form*[1]*:*

SELECT $f(A)$ FROM table
WHERE $x_1 \leq C_1 \leq y_1$ and $\cdots$ and $x_d \leq C_d \leq y_d$

where f*,* A*, and* $C_i (i \in [1, d])$ *are called aggregation function, aggregation attribute, and condition attributes, respectively, and* x_i, y_i *are in the data domain of* C_i *for each* $i \in [1, d]$*.*

For example, if a user wants to explore the relationship between product sales and customer ages, she can specify a query template like $[\texttt{SUM}(sale), age]$.

For ease of presentation, we assume that $f = \texttt{SUM}$ in later text and discuss the extension to other aggregation functions in Appendix 3.7. Moreover, we assume that the data domain of each C_i is $\mathsf{dom}(C_i) = \{1, 2, \cdots, |\mathsf{dom}(C_i)|\}$[2], and abbreviate a range query as $\texttt{SUM}(x_1 : y_1, x_2 : y_2, \cdots, x_d : y_d)$.

[1]Note that this form of queries subsumes those with other forms of range conditions, e.g., "$C = x$", "$C \geq x$", "$x \leq C < y$". This work focuses on single-table queries, but it is straightforward to extend AQP++ to handle foreign key joins using a similar idea from [47]. We defer other complex join queries to future work.

[2]If C_i does not have a natural ordering (e.g., country), we use an alphabetical ordering.

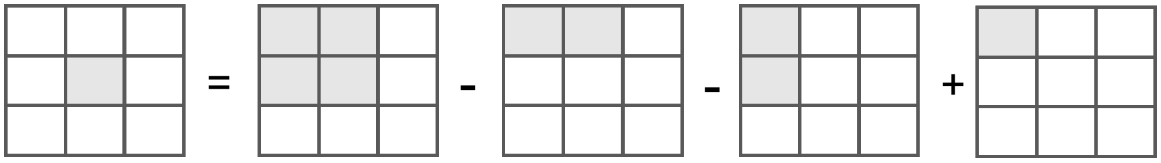


Figure 3.1: A geometric illustration of the 2-D case.

In AggPre literature [91, 61, 84, 71, 111], people often precompute a *prefix cube* (P-Cube) (or its variations) to answer range queries.

Definition 2 (Prefix Cube)**.** *Given a query template Q : $[\texttt{SUM}(A), C_1, C_2, \cdots C_d]$, the prefix cube consists of the answers to all the queries of the following form:*

$$\texttt{SUM}(1 : y_1, 1 : y_2, \cdots , 1 : y_d),$$

where $y_i \in \textsf{dom}(C_i)$ and $i \in [1, d]$.

Each answer is called a *cell* in the cube. A nice property about P-Cube is that for any range query in Q, its answer can be computed from no more than 2^d cells. For example, consider a 1D range query: $\texttt{SUM}(3 : 5)$. The answer for this query can be obtained from 2 cells, i.e., $\texttt{SUM}(1 : 5) - \texttt{SUM}(1 : 2)$. For a 2D range query, the answer can be obtained from at most 4 cells. There is a geometric illustration in Figure 3.1.

However, it is often expensive to precompute the entire P-Cube (with $\prod_{i=1}^{d} |\textsf{dom}(C_i)|$ cells). Thus, we precompute a *blocked prefix cube* (BP-Cube) [91] consisting of a small portion of the cells.

Definition 3 (Blocked Prefix Cube)**.** *Given a query template Q, let $\textsf{dom}(C_i)_{small}$ denote a subset of $\textsf{dom}(C_i)$ for each $i \in [1, d]$. The blocked prefix cube consists of the answers to all the queries of the following form:*

$$\texttt{SUM}(1 : y_1, 1 : y_2, \cdots , 1 : y_d),$$

where $y_i \in \textsf{dom}(C_i)_{small}$ and $i \in [1, d]$.

BP-Cube reduces the number of cells to be precomputed from $\prod_{i=1}^{d} |\textsf{dom}(C_i)|$ to $\prod_{i=1}^{d} |\textsf{dom}(C_i)_{small}|$. For example, consider a 2D query template Q : $[\text{SUM}(A), C_1, C_2]$, where $\textsf{dom}(C_1) = \{1, 2, \cdots , 15\}$ and $\textsf{dom}(C_2) = \{1, 2, \cdots , 8\}$. The full P-Cube contains $15 * 8 = 120$ cells, i.e., $\texttt{SUM}(1 : y_1, 1 : y_2)$ for all $y_1 \in [1, 15]$ and $y_2 \in [1, 8]$. Suppose $\textsf{dom}(C_1)_{small} = \{5, 10, 15\}$ and $\textsf{dom}(C_2)_{small} = \{4, 8\}$. Then the BP-Cube only contains $3 * 2 = 6$ cells, i.e., $\texttt{SUM}(1 : y_1, 1 : y_2)$ for all $y_1 \in \{5, 10, 15\}$ and $y_2 \in \{4, 8\}$.

Aggregate Identification. We now start defining the aggregate-identification problem. Let P denote a BP-Cube, i.e.,

$$P = \big\{\texttt{SUM}(1 : y_1, 1 : y_2, \cdots , 1 : y_d) \mid \text{for all } y_i \in \textsf{dom}(C_i)_{small} \text{ and } i \in [1, d]\big\}. \quad (3.1)$$

Note that although only P is precomputed, due to the properties of the BP-Cube, we can assume that the answers to all the queries in P^+ are available.

$$P^+ = P \ \cup \ \{\phi\} \cup \Big\{ \texttt{SUM}(x_1 + 1 : y_1,\ x_2 + 1 : y_2,\ \cdots,\ x_d + 1 : y_d) \mid \text{for all } x_i, y_i \in \mathsf{dom}(C_i)_{small} \text{ and } i \in [1, d] \Big\}. \quad (3.2)$$

For example, suppose $\mathsf{dom}(C_1)_{small} = \{4, 6\}$. Then, we have $P = \{\texttt{SUM}(1:4), \texttt{SUM}(1:6)\}$, and $P^+ = \{\texttt{SUM}(1:4), \texttt{SUM}(1:6), \texttt{SUM}(5:6), \phi\}$, where $\texttt{SUM}(5:6)$ can be obtained from $\texttt{SUM}(1:6) - \texttt{SUM}(1:4)$ and ϕ is an empty query with an always-false condition.

Given a user query q and a sample S, for each $pre \in P^+$, AQP++ can return an approximate answer along with a confidence interval (see Section 3.4.2 for more detail about this). For example, suppose $q : \texttt{SUM}(1:4)$. Imagine AQP++ leverages the precomputed query $pre : \texttt{SUM}(2:4)$ and returns 1000 ± 5 (confidence level: 95%). This result indicates that the true answer of q is in the range of $[995, 1005]$ with 95% probability. The larger the width of the confidence interval, the less accurate the approximate answer. For this reason, we define the *query error* of q w.r.t. pre, denoted by $error(q, pre)$, as half the width of the confidence interval. For the above example, the confidence interval has the width of 10, thus $error(q, pre) = 5$.

Once a BP-Cube P is precomputed, since any value in P^+ can be leveraged to answer a user query, we aim to select the best one with the minimum query error. We denote the minimum query error w.r.t. P by $error(q, P) = \min_{pre \in P^+} error(q, pre)$. Problem 1 formally defines the problem.

Problem 1 (Aggregate Identification)**.** *Given a user query q, a sample S, and a BP-Cube P, the goal of aggregate identification is to identify the best value in P^+ such that the query error is minimized:*

$$\arg\min_{pre \in P^+} error(q, pre)$$

Consider a 1D example. Given a user query $q : \texttt{SUM}(2:5)$ and a BP-Cube $P = \{\texttt{SUM}(1:4), \texttt{SUM}(1:6)\}$, there are four precomputed values in $P^+ = \{\texttt{SUM}(1:4), \texttt{SUM}(1:6), \texttt{SUM}(5:6), \phi\}$. The aggregate-identification problem aims to identify the best value among the four values and use it to estimate the answer to q.

Aggregate Precomputation. Next, we define the aggregate-precomputation problem. Given a space budget, we want to find the best BP-Cube that satisfies the space budget. Let $|P|$ denote the number of cells in a BP-Cube P and k denote a threshold bounding $|P|$. Given a threshold k, there are a lot of different ways to construct a BP-Cube such that $|P| \le k$. For example, suppose $k = 6$ and $d = 2$. The shapes of BP-Cubes can be 1×6, 2×3, 3×2 or 6×1. For each possible shape, e.g., 2×3, a BP-Cube can be constructed by choosing any 2 values from $\mathsf{dom}(C_1)$ and any 3 values from $\mathsf{dom}(C_2)$.

To decide which one is the best, we define *query-template error* to quantify the benefit of each BP-Cube and return the one with the minimum query-template error. Given a query template Q and a BP-Cube P, since a user might issue any query in Q, we define the *query-template error* w.r.t P as $error(Q, P) = \max_{q \in Q} error(q, P)$, which is the maximum query error over all possible user queries. We choose this error metric because it is more beneficial to reduce the errors of highly inaccurate queries rather than the ones who have already gotten very accurate results. Certainly, there are many other ways to define a query-template error. In the experiments, we show that our aggregate-precomputation approach, which is designed to optimize the maximum error, can also significantly reduce other types of errors, such as average error and median error. Problem 2 defines the aggregate-precomputation problem.

Problem 2 (Aggregate Precomputation)**.** *Given a query template Q, a sample S, and a threshold k, the goal of aggregate precomputation is to determine the best BP-Cube P such that $|P| \leq k$ and the query-template error is minimized:*

$$\arg\min_{P} error(Q, P) \;\; s.t. \;\; |P| \leq k$$

Consider a 1D example. Given a threshold k = 2 and a query template $Q : [\texttt{SUM}(A), C_1]$, where $\mathsf{dom}(C_1) = \{1, 2, \cdots, 5\}$, the aggregate-precomputation problem aims to determine the best BP-Cube $P = \{\texttt{SUM}(1 : t_1), \texttt{SUM}(1 : t_2)\}$ with the two values t_1 and t_2 chosen from $\mathsf{dom}(C_1)$ such that $error(Q, P)$ is minimized. For multiple-dimensional cases, the problem becomes even more challenging because we also need to determine the shape of the best BP-Cube, e.g., 2×3 or 3×2.

3.4 From AQP to AQP++

In this section, we first provide some background knowledge about AQP and then present the AQP++ framework.

3.4.1 Sampling-based AQP

AQP's mathematical foundations are built on the sampling and estimation theories [73].
Result Estimation. Given a large relational table $\mathcal{D}$ and a random sample $\mathcal{S}$ of the table, for certain aggregation queries q, their answers can be estimated based on the sample, i.e.,

$$q(\mathcal{D}) \approx \hat{q}(\mathcal{S}), \tag{3.3}$$

or simply $q \approx \hat{q}$ if the context is clear. The supported aggregation queries are of this form:

```
SELECT f(A) FROM D WHERE Condition.
```

Please note that `Condition` can be ***any*** function that takes a record as input and returns True or False (e.g., `age > 30 and country = "USA"`, `tweet like "%sigmod%"`).

Note that f cannot be an arbitrary aggregation function (e.g., min, max). Typical aggregation functions include AVG, SUM, COUNT, and VAR. A recent paper shows that some kinds of User Defined Functions (UDFs) can be supported as well [46].

Confidence Interval. Along with an estimated result, AQP often returns a confidence interval to quantify the estimation uncertainty. The confidence interval is an interval estimate of the true value. For example, a 95% confidence interval $\hat{q} \pm \epsilon$ means that the true value is within this range with 95% probability. In AQP, there are two kinds of approaches to computing a confidence interval. The first one is an analytical approach, aiming to derive a closed-form confidence interval often based on the Central Limit Theorem. The limitation of this approach is the lack of generality. Each query has its own form of confidence interval. Example 2 shows how to compute the closed-form confidence interval for a SUM query.

Example 2. *Suppose we want to use AQP to estimate the answer along with the confidence interval for the following query:*

$$q(\mathcal{D}) \texttt{ = SELECT SUM(}A\texttt{) FROM } \mathcal{D} \texttt{ WHERE } C \ \geq \ 0$$

To facilitate the calculation, we first rewrite it as follows:

$$q(\mathcal{D}) \texttt{ = SELECT SUM(}A \cdot \mathsf{cond}(C \geq \ 0)\texttt{) FROM } \mathcal{D}$$

where $\mathsf{cond}(C \geq 0)$ *returns 1 if* $C \geq 0$ *holds; 0, otherwise.*

For ease of presentation, we denote $A \cdot \mathsf{cond}(C \geq 0)$ *by A', the table size* $|\mathcal{D}|$ *by* N*, and the sample size* $|\mathcal{S}|$ *by* n*. Then, we can obtain the estimated answer to q:*

$$\hat{q}(\mathcal{S}) \texttt{ = SELECT } N \cdot \frac{\texttt{SUM}(A')}{n} \texttt{ FROM } \mathcal{S}$$

Based on the Central Limit Theorem, we can derive that the closed-form confidence interval of the query is $\hat{q} \pm \epsilon$*, where*

$$\epsilon \texttt{ = SELECT } \lambda \cdot N\sqrt{\frac{\texttt{VAR}(A')}{n}} \texttt{ FROM } S$$

Here λ *is a parameter determined by a confidence level. For example,* $\lambda = 1.96$ *for a 95% confidence interval and* $\lambda = 2.576$ *for a 99% confidence interval.*

It may be very hard to get a closed-form confidence interval for more complex queries. Thus AQP often computes an empirical confidence interval using bootstrap. The approach generates a set of resamples, $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_m$, of the original sample $\mathcal{S}$, and estimates the query answer using each resample. The obtained answers, $\hat{q}(\mathcal{S}_1), \hat{q}(\mathcal{S}_2), \cdots, \hat{q}(\mathcal{S}_m)$, form an estimate of the distribution of $q(\mathcal{D})$, from which we can compute a confidence interval. This is a more general approach, but a naive implementation will suffer from high computational cost because it needs to generate m resamples and then run queries on each resamples. Some sophisticated approaches were proposed to overcome the limitation [138, 177].

3.4.2 AQP++ Framework

The AQP++ framework is tailored to connect AQP with AggPre for interactive analytics. In this section, we first answer two fundamental questions about AQP++: (1) *How does AQP++ estimate a query result?* (2) *How does AQP++ compute a confidence interval?*

Result Estimation

Unlike AQP that uses a sample to *directly* estimate the answer to a user query (see Equation 3.3), AQP++ seeks to use a sample to estimate the difference of the query result from a pre-computed aggregate value. Let q denote a *user query* and pre denote a *precomputed aggregate query.*

q: SELECT $f(A)$ FROM $\mathcal{D}$ WHERE Condition$_1$
pre: SELECT $f(A)$ FROM $\mathcal{D}$ WHERE Condition$_2$

AQP++ estimates the difference of their query results as follows:

$$q(\mathcal{D}) - pre(\mathcal{D}) \approx \hat{q}(\mathcal{S}) - \hat{pre}(\mathcal{S})$$

Since $pre(\mathcal{D})$ has been precomputed (i.e., a constant), we have

$$q(\mathcal{D}) \approx pre(\mathcal{D}) + \big(\hat{q}(\mathcal{S}) - \hat{pre}(\mathcal{S})\big) \tag{3.4}$$

To compute Equation 3.4, AQP++ first employs AQP (Equation 3.3) to estimate the user query's answer $\hat{q}(\mathcal{S})$, and then estimate the precomputed query's answer $\hat{pre}(\mathcal{S})$. Finally, it plugs the two values into Equation 3.4 to obtain the final estimated answer.

Example 3. *This example illustrates how to use AQP++ to estimate the answer to the same query as Example 2:*

$$q(\mathcal{D}) \texttt{ = SELECT SUM}(A) \texttt{ FROM } \mathcal{D} \texttt{ WHERE } C \geq 0$$

Suppose the following aggregate query has been precomputed:

$$pre(\mathcal{D}) \texttt{ = SELECT SUM}(A) \texttt{ FROM } \mathcal{D} \texttt{ WHERE } C > 0$$

AQP++ first uses AQP (Equation 3.3) to estimate the answers to q and pre, respectively.

$$\begin{aligned} \hat{q}(\mathcal{S}) &\texttt{ = SELECT } N \cdot \frac{\texttt{SUM}(A \cdot \mathsf{cond}(C \geq 0))}{n} \texttt{ FROM } \mathcal{S} \\ \hat{pre}(\mathcal{S}) &\texttt{ = SELECT } N \cdot \frac{\texttt{SUM}(A \cdot \mathsf{cond}(C > 0))}{n} \texttt{ FROM } \mathcal{S} \end{aligned}$$

Then, it plugs $pre(\mathcal{D})$, $\hat{q}(\mathcal{S})$, and $\hat{pre}(\mathcal{S})$ into Equation 3.4 to get the estimated answer to q.

Unification. AQP++ connects AQP with AggPre using Equation 3.4. Interestingly, the equation shows that AQP and AggPre are only the special cases of AQP++ when no aggregate is precomputed, and all aggregates are precomputed, respectively.

1. *AQP++ subsumes AQP*. Define ϕ as an *empty query*, whose condition is always false.

   $$\phi \texttt{ = SELECT SUM}(A) \texttt{ FROM } \mathcal{D} \texttt{ WHERE False}^{3}.$$

   Suppose no aggregate is precomputed, i.e., $pre = \phi$. In this case, we have $pre(\mathcal{D}) = \hat{pre}(\mathcal{S}) = 0$. Thus, Equation 3.4 is reduced to $q(D) \approx \hat{q}(S)$, which returns the same result as AQP.

2. *AQP++ subsumes AggPre*. Suppose all aggregates are precomputed, i.e., $pre = q$. In this case, we have $\hat{pre}(\mathcal{S}) = \hat{q}(\mathcal{S})$. Thus, Equation 3.4 is reduced to $q(\mathcal{D}) \approx pre(\mathcal{D})$, which returns the same result as AggPre.

Generality. The AQP++ framework is very general. Firstly, it works for any aggregate function that AQP can support such as SUM, COUNT, AVG, and VAR, and some UDFs (see Lemma 1). This can be easily proved based on Equation 3.4 because $q(\mathcal{D})$ only depends on $pre(\mathcal{D})$, $\hat{q}(\mathcal{S})$, and $\hat{pre}(\mathcal{S})$, where $pre(\mathcal{D})$ has been precomputed, and $\hat{q}(\mathcal{S})$ and $\hat{pre}(\mathcal{S})$ can be obtained using AQP (since AQP can support their aggregation function).

Lemma 1. *For any aggregation function f, if AQP can answer the queries of the form:* `SELECT f(A) FROM D WHERE Condition`*, AQP++ can answer the queries as well.*

Proof. The proofs to the lemmas and theorems of this work can be found in Appendix A.1. □

Furthermore, for any aggregate function, if AQP has an unbiased estimator, AQP++'s estimator is also unbiased.

Lemma 2. *For any aggregation function f, if AQP can unbiasedly estimate the queries of the form:* `SELECT f(A) FROM D WHERE Condition`*, the answers that AQP++ returns are also unbiased.*

Lastly, AQP++ can be easily extended to support many existing optimization techniques that were proposed for AQP, such as stratified sampling [43, 62, 50, 63, 64, 153, 47, 83] and auxiliary indices [62, 78]. These optimization techniques aim to improve the estimation quality of Equation 3.3. Since Equation 3.4 is obtained by treating Equation 3.3 as a black box, the optimization techniques work for AQP++ as well. For example, consider stratified sampling optimization. Unlike uniform sampling, where each row has the same probability of being sampled, stratified sampling divides data into a set of subgroups and tends to apply a higher sampling rate to smaller subgroups. This was shown to be an effective sampling approach for skewed group-size distributions. Let S_{op} be a stratified sample of data. Then, we have an optimized AQP++ framework:

$$q(\mathcal{D}) \approx pre(\mathcal{D}) + \big(\hat{q}(\mathcal{S}_{op}) - \hat{pre}(\mathcal{S}_{op})\big), \tag{3.5}$$

[3]Note that AQP++ also works when q and pre have different aggregation functions.

where $\hat{q}(\mathcal{S}_{op})$ and $\hat{pre}(\mathcal{S}_{op})$ apply AQP to the stratified sample to estimate the answers to q and pre, respectively.

Confidence Interval

Like AQP, AQP++ also has two kinds of approaches to compute a confidence interval for an estimated answer. The analytical one needs to manually calculate a closed-form confidence interval for each query type. Example 4 illustrates how to compute a confidence interval for a SUM query.

Example 4. *Continuing Example 3, suppose we want to compute a confidence interval for $pre(\mathcal{D})+(\hat{q}(\mathcal{S})-\hat{pre}(\mathcal{S}))$. Since $pre(\mathcal{D})$ is a constant, we only need to compute the confidence interval for $\hat{q}(\mathcal{S})-\hat{pre}(\mathcal{S})$, i.e.,*

$$\texttt{SELECT } N \cdot \frac{\texttt{SUM}(A\cdot\mathsf{cond}(C=0))}{n} \texttt{ FROM } S.$$

Following the idea of Example 2, we obtain the confidence interval $\hat{q} \pm \epsilon$, where

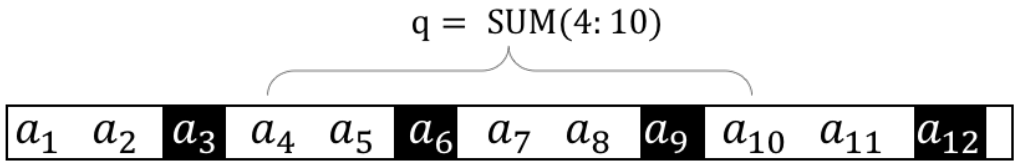
$$\epsilon \texttt{ = SELECT } \lambda \cdot N\sqrt{\frac{\texttt{VAR(A}\cdot\mathsf{cond}(C=0))}{n}} \texttt{ FROM } S.$$

When it is hard to derive closed-form confidence intervals for more complex queries, AQP++ can use the bootstrap to compute their empirical confidence intervals. It first generates a set of resamples, $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_m$. Then, for each resample $\mathcal{S}_i$ $(i \in [1, m])$, it computes $pre(\mathcal{D})+(\hat{q}(\mathcal{S}_i)-\hat{pre}(\mathcal{S}_i))$. Please note that this step is different from AQP (which computes $\hat{q}(\mathcal{S}_i)$ instead). The computed results form an estimate of the distribution of $q(\mathcal{D})$, from which we derive a confidence interval.

Back of the envelope analysis. We investigate why AQP++ may produce more accurate answers (i.e., tighter confidence intervals) compared to AQP. One may be tempted to think this is impossible because AQP++ has to estimate *two* random variables and then get their difference, which should have introduced more error than AQP, which only needs to estimate *one*. However, this analysis ignores the correlation between the two random variables. For AQP++, the variance of its estimator (involving two random variables) is:

$$\mathsf{Var}\Big(\hat{q}-\hat{pre}\Big) = \mathsf{Var}(\hat{q}) + \mathsf{Var}(\hat{pre}) - 2\mathsf{Cov}(\hat{q}, \hat{pre})$$

For AQP, the variance of its estimator is $\mathsf{Var}(\hat{q})$. By comparing them, we can see that if $\mathsf{Var}(\hat{pre}) < 2\mathsf{Cov}(\hat{q}, \hat{pre})$, AQP++ can have a smaller variance than AQP. That is, AQP++ tends to return a more accurate result in this situation. To further elaborate on the situation, we know that $\mathsf{Cov}(\hat{q}, \hat{pre})$ depends on the degree of the correlation between a user query result and a pre-computed aggregate. If they are highly correlated, $\mathsf{Cov}(\hat{q}, \hat{pre})$ becomes very large, thus AQP++ is more likely to return a more accurate answer. We also validate this interesting phenomenon in the experiments.


q = SUM(4: 10)
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12}

$$P = \{\text{SUM}(1{:}\ 3), \text{SUM}(1{:}\ 6), \text{SUM}(1{:}\ 9), \text{SUM}(1{:}\ 12)\}$$

$$P^{+} = \{\text{SUM}(1{:}\ 3), \text{SUM}(1{:}\ 6), \text{SUM}(1{:}\ 9), \text{SUM}(1{:}\ 12), \text{SUM}(4{:}\ 6), \text{SUM}(4{:}\ 9), \text{SUM}(4{:}\ 12), \text{SUM}(7{:}\ 9), \text{SUM}(7{:}\ 12), \text{SUM}(10{:}\ 12), \emptyset\}$$

$$P^{-} = \{\text{SUM}(4{:}\ 9), \text{SUM}(4{:}\ 12), \text{SUM}(7{:}\ 9), \text{SUM}(7{:}\ 12), \emptyset\}$$

Figure 3.2: An illustration of P, P^+, and P^- for the 1D case.

3.5 Aggregate Identification

With the new AQP++ framework, we now study the two problems defined in Section 4.2. This section studies the aggregate-identification problem. We first consider a simplified problem setting and present an optimal solution. We then extend the solution to the general setting.

3.5.1 Optimal Solution

We assume that (1) Q is one-dimensional (i.e., [SUM(A), C]); (2) A and C are independent. For ease of presentation, we denote a relational table by $\mathcal{D} = [a_1, a_2, \cdots, a_N]$, which is a list of attribute values of A ordered by C. Since A and C are independent, the list can be thought of as being randomly shuffled. Each user query is denoted by $\text{SUM}(x : y) = \sum_{x \le i \le y} a_i$. We denote a BP-Cube by $P = \{\text{SUM}(1 : t) = \sum_{1 \le i \le t} a_i \text{ for all } t \in \mathsf{dom}(C)_{small}\}$, and we call each $t \in \mathsf{dom}(C)_{small}$ *a partition point.*

Given a user query q, to identify the best aggregate value for q, the brute-force approach needs to compute the query error w.r.t *every pre* $\in P^+$ and return the one with the minimum error. Since $|P^+|$ can be very large, this approach is prohibitively expensive. The key observation of our approach is that a user query can benefit more from a correlated aggregate query than an uncorrelated one. For example, consider a user query SUM$(4 : 10)$. Suppose both SUM$(4 : 9)$ and SUM$(1 : 3)$ have been precomputed. Without the need to compute the actual query error w.r.t them, we can immediately deduce that SUM$(4 : 9)$ is preferable because it is highly correlated to the user query while SUM$(1:3)$ tells us nothing about the user query.

Based on this observation, given a user query SUM$(x:y)$, we can prove that only five aggregate values in P^+ need to be considered. We call them *candidate aggregate values*, denoted by

$$P^- = \big\{\mathrm{SUM}(l_x+1:l_y), \mathrm{SUM}(l_x+1:h_y), \mathrm{SUM}(h_x+1:l_y), \mathrm{SUM}(h_x+1:h_y)\big\} \cup \big\{\phi\big\}, \quad (3.6)$$

where l_x (h_x) is the first partition point that is lower (higher) than x; l_y (h_y) is the first partition point that is lower (higher) than y. Intuitively, x falls between partition points l_x and h_x, and y falls between partition points l_y and h_y. Based on the four partition points, we find the four most correlated aggregate queries to the user query. We also add ϕ into P^- to handle some special cases, e.g., x and y fall between the same two partition points (i.e., $l_x = l_y$). Lemma 3 proves the correctness of this solution.

Lemma 3. *Given $\mathcal{D} = [a_1, \cdots, a_N]$, a query template Q, and a* BP-Cube *P, we have:*

$$\min_{pre \in P^+} error(q, pre) = \min_{pre \in P^-} error(q, pre).$$

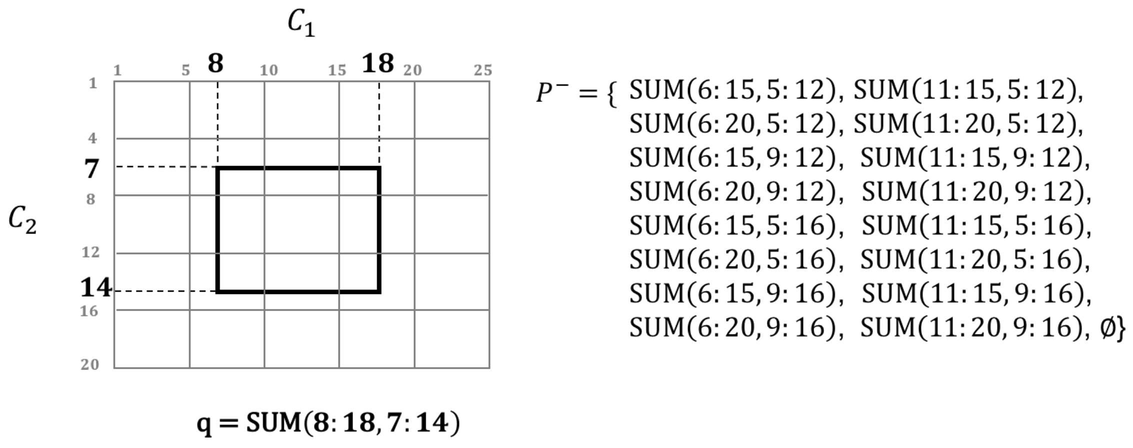
For example, consider the BP-Cube P and the available aggregate values P^+ in Figure 3.2. Given a user query $q = \mathrm{SUM}(4:10)$, we want to identify the best value from P^+ for the query. Based on Lemma 3, we only need to consider five values. To get the five values, we can see that $x = 4$ falls between the precomputed points of 3 and 6, thus $l_x = 3$ and $h_x = 6$; $y = 10$ falls between the precomputed points of 9 and 12, thus $l_x = 9$ and $h_x = 12$. Based on Equation 3.6, we obtain $P^- = \{\mathrm{SUM}(4:9), \mathrm{SUM}(4:12), \mathrm{SUM}(7:9), \mathrm{SUM}(7:12), \phi\}$.

3.5.2 Aggregate-Identification Approach

Lemma 3 significantly reduces the number of aggregate values that need to be considered for answering a user query. While we can only prove its correctness under certain assumptions, it inspires us to develop an efficient heuristic approach for the general setting.

The key idea of our approach is to quickly identify a small number of candidate aggregate values from P^+ and then examine which one is the best among them. Similarly, we denote the candidate aggregate values by P^-. Equation 3.6 shows the computation of P^- for the 1D case. We now extend it to the d-dimensional case. Given a user query $q = \mathrm{SUM}(x_1 : y_1, x_2 : y_2, \cdots, x_d : y_d)$, suppose that for each $i \in [1, d]$, x_i falls between l_{x_i} and h_{x_i}, and y_i falls between l_{y_i} and h_{y_i}. We define P^- w.r.t q as

$$P^- = \Big\{\mathrm{SUM}(u_1+1:v_1,\ u_2+1:v_2,\ \cdots,\ u_d+1:v_d) \mid u_i \in \{l_{x_i}, h_{x_i}\}, v_i \in \{l_{y_i}, h_{y_i}\} \text{ for each } i \in [1,d]\Big\} \cup \Big\{\phi\Big\} \quad (3.7)$$



$$\begin{aligned}P^- = \{\ &\mathrm{SUM}(6{:}15, 5{:}12),\ \mathrm{SUM}(11{:}15, 5{:}12),\\ &\mathrm{SUM}(6{:}20, 5{:}12),\ \mathrm{SUM}(11{:}20, 5{:}12),\\ &\mathrm{SUM}(6{:}15, 9{:}12),\ \mathrm{SUM}(11{:}15, 9{:}12),\\ &\mathrm{SUM}(6{:}20, 9{:}12),\ \mathrm{SUM}(11{:}20, 9{:}12),\\ &\mathrm{SUM}(6{:}15, 5{:}16),\ \mathrm{SUM}(11{:}15, 5{:}16),\\ &\mathrm{SUM}(6{:}20, 5{:}16),\ \mathrm{SUM}(11{:}20, 5{:}16),\\ &\mathrm{SUM}(6{:}15, 9{:}16),\ \mathrm{SUM}(11{:}15, 9{:}16),\\ &\mathrm{SUM}(6{:}20, 9{:}16),\ \mathrm{SUM}(11{:}20, 9{:}16),\ \emptyset\}\end{aligned}$$

Figure 3.3: An illustration of P^- for the 2D case.

For example, consider a 2D BP-Cube in Figure 3.3. The first dimension has the partition points of $\mathsf{dom}(C_1)_{small} = \{1, 5, 10, 15, 20, 25\}$; the second dimension has the partition points of $\mathsf{dom}(C_2)_{small} = \{1, 4, 8, 12, 16, 20\}$. Given a user query $q = \texttt{SUM}(8 : 18, 7 : 14)$, for the first dimension C_1, we can see that $x_1 = 8$ falls between 5 and 10, thus $l_{x_1} = 5$ and $h_{x_1} = 10$; $y_1 = 18$ falls between 15 and 20, thus $l_{y_1} = 15$ and $h_{y_1} = 20$. Similarly, for the second dimension, we have $l_{x_2} = 4$ and $h_{x_2} = 8$ for $x_2 = 7$; we have $l_{y_2} = 12$ and $h_{y_2} = 16$ for $y_2 = 14$. Based on Equation 3.7, we obtain P^- consisting of the 17 aggregate values shown on the right side of the figure.

The size of P^- is independent of the BP-Cube size. For each dimension, there are four possible cases, i.e., $\{l_{x_i}, h_{x_i}\} \times \{l_{y_i}, h_{y_i}\}$. Thus, we have $|P^-| = 4^d + 1$. For example, $|P^-| = 4^1 + 1 = 5$ for the 1D case and $|P^-| = 4^2 + 1 = 17$ for the 2D case. Furthermore, we can quickly identify P^-. Let $k_i = |\mathsf{dom}(C_i)_{small}|$. Every dimension only needs $\mathcal{O}(\log k_i)$ time to search for the partition points, thus all dimensions need $\mathcal{O}(\sum_i \log(k_i)) = \mathcal{O}(\log k)$ time. To generate $4^d + 1$ queries based on the partition points, we need $\mathcal{O}(4^d)$ time. Thus, the total time complexity is $\mathcal{O}(\log k + 4^d)$.

Once P^- is obtained, we then need to decide which one in P^- should be finally identified. We adopt a subsampling based approach. The idea is to create a subsample of $\mathcal{S}$, and use the subsample to estimate the query error w.r.t. each aggregate value in P^-. Specifically, given a user query, for each precomputed aggregate value $pre \in P^-$, we compute the confidence interval of the user query w.r.t. pre and select the one with the smallest confidence interval. The subsampling rate is a parameter that can balance the trade-off between the effectiveness and efficiency of aggregate identification. In the experiments, we set it to less than $\frac{1}{4^d}$ to ensure that the overhead added is smaller than the actual query processing time.

3.6 Aggregate Precomputation

We now study the aggregate-precomputation problem. There are two major challenges. One is how to determine the BP-Cube's shape. That is, given a threshold k, we need to assign a number k_i to each dimension such that $\prod_{i=1}^{d} k_i \leq k$. The total number of possible assignments can be quite large. The other challenge is how to decide which k_i points should be chosen from $\mathsf{dom}(C_i)$ for each $i \in [1, d]$. Again, there are a large number of different choices, $\binom{|\mathsf{dom}(C_i)|}{k_i}$, and a brute-force approach does not work.

In this section, we first explore the 1-dimensional case and then extend it to multidimensional cases.

3.6.1 One-Dimensional Query Template

For the one-dimensional case, since the BP-Cube's shape has only one possibility, the only challenge left is how to choose the best k points, $1 \leq t_1 < t_2 < \cdots < t_k = |\mathsf{dom}(C)|$, from $\mathsf{dom}(C)$ such that the query-template error is minimized[4]. The most natural idea, called *equal-partition scheme*, is to partition $\mathsf{dom}(C)$ into k equal parts. But, this idea ignores two important factors.

- *Data Distribution.* If C does not follow a uniform distribution (i.e., some values appear more frequently than others), the equal-partition scheme is often not feasible. This is, we cannot use range queries with conditions over C to partition A equally. For example, consider the relational table (with attributes A and C) in Figure 3.4(a). The only way to partition the data is shown in the figure, which is not an equal-partition scheme.

- *Attribute Correlation.* If A and C are correlated when sorting A by C, this process is not equivalent to a random shuffle of A. Thus, the variances of different parts of A may differ greatly. For example, consider the relational table in Figure 3.4(b). When $1 \leq C \leq 4$, A is always equal to 0; when $5 \leq C \leq 8$, A follows a normal distribution with a large variance. Since a larger variance leads to a higher query error, it might be better to choose more points from the second half of A rather than adopt an equal-partition scheme.

In the following, we first make some assumptions about data distribution and attribute correlation and prove that the equal-partition scheme is optimal under these assumptions. Then, we relax the assumptions and propose an adaptive approach for the general setting.

[4]We compute the sum of all the values in each aggregation attribute because these sum values are independent of condition attributes and can be reused across query templates. Therefore, we assume $t_k = |\mathsf{dom}(C)|$

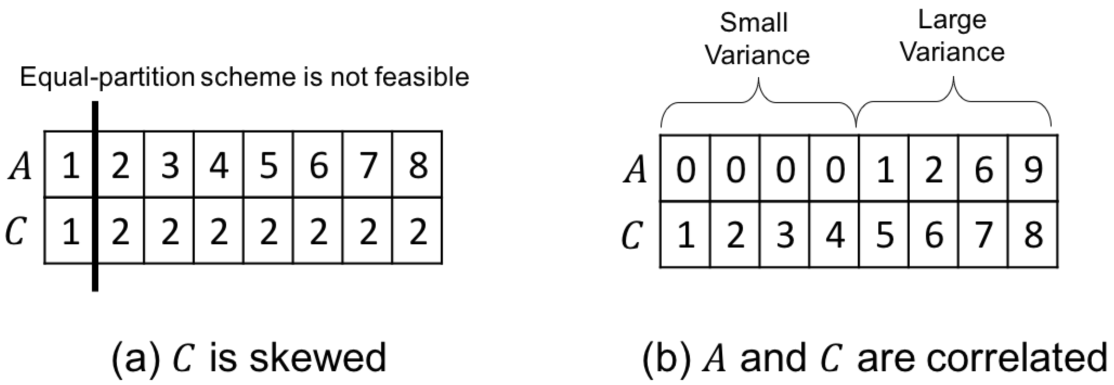


Figure 3.4: (a) The equal-partition scheme is not feasible; (b) The equal-partition scheme is not optimal.

Optimal Partition Scheme

We assume that (1) C has no duplicate values; (2) A and C are independent. The first (resp. second) assumption removes the impact of the data distribution of C (resp. the correlation between attributes A and C) on the optimal partition scheme. Similar to Section 3.5.1, we denote a relational table by $\mathcal{D} = \{a_1, a_2, \cdots, a_N\}$, which is the list of attribute values of A ordered by C. Assumption 1 suggests that any sub-list of $\mathcal{D}$ can be precomputed (without being constrained by the skewed distribution of C); Assumption 2 means that $\mathcal{D}$ can be thought of as being randomly shuffled.

We can prove that the equal-partition scheme is optimal under Assumptions 1 and 2. The corresponding BP-Cube is denoted by[5]

$$P_{eq} = \{\text{SUM}(1 : \frac{i}{k}N) \mid i = 1, 2, \cdots k\}.$$

The proof's basic idea is that in Lemma 4, we compute the query-template error, $error(Q, P_{eq})$, w.r.t. the equal-partition scheme; in Lemma 5, we prove that for any other partition scheme, the resulting query-template error cannot be smaller than $error(Q, P_{eq})$. Hence, P_{eq} is optimal since it has the minimum query-template error.

Lemma 4. *Given $\mathcal{D} = \{a_1, a_2, \cdots, a_N\}$, a query template Q, and a threshold k, the query-template error of Q w.r.t P_{eq} is*

$$error(Q, P_{eq}) = \lambda N \sqrt{\frac{\sigma_{eq}^2}{n}},$$

where $\sigma_{eq}^2 = \frac{1}{k}E[\mathcal{D}^2] - \frac{1}{k^2}(E[\mathcal{D}])^2$.

[5]Here, we assume $N\%k = 0$. Otherwise, we will choose $\lceil \frac{i}{k}N \rceil$ for $i \in [1, N\%k]$, and $\lfloor \frac{i}{k}N \rfloor$ for $i \in (N\%k, k]$. The proof can be extended to this case.

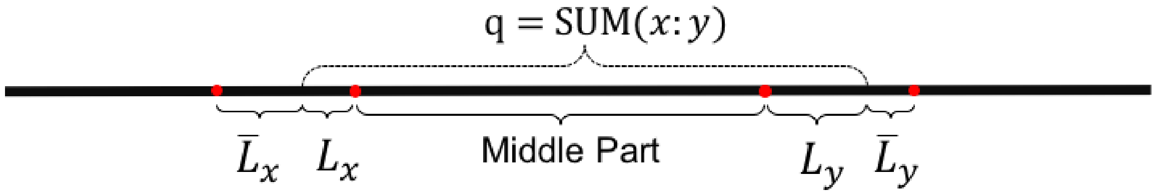


Figure 3.5: An illustration for notations of $L_x, \bar{L}_x, L_y$, and $\bar{L}_y$

Lemma 4 indicates that the query-template error decreases at a rate of $\mathcal{O}(\frac{1}{\sqrt{k}})$. This is a fascinating result because it shows that the query-template error can be dramatically reduced with only a small k (i.e., the BP-Cube size). For example, if $k = 100$, the error can be reduced by about ten times.

Lemma 5 proves that $error(Q, P_{eq})$ is minimum.

Lemma 5. *Given $\mathcal{D} = \{a_1, a_2, \cdots, a_N\}$, a query template Q, and a threshold k, if $P \neq P_{eq}$, then $error(Q, P) \geq error(Q, P_{eq})$.*

It is easy to prove Theorem 1 based on Lemmas 4 and 5.

Theorem 1. *Given $\mathcal{D} = \{a_1, a_2, \cdots, a_N\}$, a query template Q, and a threshold k, P_{eq} is an optimal BP-Cube.*

For example, suppose $\mathcal{D} = \{a_1, a_2, \cdots, a_{12}\}$ and $k = 4$. Based on Theorem 1, we obtain the optimal BP-Cube $P_{eq} = \{\text{SUM}(1:3), \text{SUM}(1:6), \text{SUM}(1:9), \text{SUM}(1:12)\}$.

An Adaptive Approach Based on Hill Climbing

The optimal partition scheme requires two assumptions which may not hold in practice. In this section, we propose a hill-climbing-based algorithm that can adaptively adjust the partition scheme based on the actual data distribution and attribute correlation.

Algorithm Overview. The algorithm starts with an initial BP-Cube and then attempts to improve it by moving a *single* partition point from one place to another. If the change leads to a better BP-Cube, the change is made, and the iterative process is repeated; otherwise, the algorithm is terminated. To make the algorithm work, we need to address three problems: (1) how to find an initial BP-Cube; (2) how to evaluate the effectiveness of a BP-Cube (in order to know whether the change leads to a better BP-Cube); (3) how to adjust a BP-Cube (i.e., decide which partition point should be moved away and where it should move to).

(1) Initialization. A poor initialization may not only hurt the efficiency of an optimization algorithm but also lead to a local optimum that is far from the global optimum. We use P_{eq} as an initialization because (1) it has been proven to be optimal in some situations, and (2) it avoids ending up with a solution that is even worse than the naive equal partitioning.

However, P_{eq} may not always be feasible (due to the skewed distribution of C). If a partition point (in P_{eq}) is not feasible, we will choose its closest feasible point to replace it. For example, in Figure 3.4(a), since the middle point (4th point) is not feasible, its closest feasible partition point (6th point) will be chosen. Accordingly, the initial BP-Cube is $P = \{\text{SUM}(1:2), \text{SUM}(1:3)\}$.

(2) Evaluation. The most naive way to evaluate the effectiveness of a BP-Cube is to adopt query-template error because this is the ultimate optimization objective. But, when the assumption that A and C are independent does not hold, we have not found an efficient way to compute it without enumerating $\binom{|\mathsf{dom}(C)|+1}{2}$ possible user queries. To address this challenge, we seek to find an upper bound of query-template error. It turns out that the upper bound can not only be efficiently computed (in linear time) but also lead to a robust solution (since it bounds the worst case).

Recall that the query-template error is defined as $error(Q, P) = \max_{q \in Q} error(q, P)$. We first give the upper bound of $error(q, P)$, and then we present an efficient linear algorithm to compute the upper bound of $error(Q, P)$.

To get the upper bound of $error(q, P)$, consider a user query $q = \text{SUM}(x : y)$ in Figure 3.5. The red dots are the partition points near x or y. We can see that the middle part of q has been precomputed, so we only need to estimate $L_x + L_y$. For L_x, since $L_x + \bar{L}_x$ has been precomputed, we can estimate L_x in two ways. One is to directly estimate L_x, and the other is to estimate the complement $\bar{L}_x$. We try both ways and choose the one with a smaller error, i.e., $\min\left\{\frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(A_{L_x})}, \frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(A_{\bar{L}_x})}\right\}$, where $A_{L_x} = A \cdot \mathsf{cond}(C \in L_x)$ and $A_{\bar{L}_x} = A \cdot \mathsf{cond}(C \in \bar{L}_x)$. Similarly, for L_y, the estimation error is $\min\left\{\frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(A_{L_y})}, \frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(A_{\bar{L}_y})}\right\}$. We obtain the upper bound of $error(q, P)$ by adding them up.

Lemma 6. *Given $\mathcal{D}$, a query template Q, and a BP-Cube P, for any $q \in Q$, we have $error(q, P) \leq$*

$$\frac{\lambda N}{\sqrt{n}} \cdot \min\left\{\sqrt{Var(A_{L_x})}, \sqrt{Var(A_{\bar{L}_x})}\right\} + \frac{\lambda N}{\sqrt{n}} \cdot \min\left\{\sqrt{Var(A_{L_y})}, \sqrt{Var(A_{\bar{L}_y})}\right\}.$$

To get the upper bound of $error(Q, P)$, we first compute $error_i = \frac{\lambda N}{\sqrt{n}} \min\left\{\sqrt{\text{Var}(A_{L_i})}, \sqrt{\text{Var}(A_{\bar{L}_i})}\right\}$ for every point $i \in [1, N]$, and then pick up two points $i_1, i_2 \in [1, N]$ where $error_{i_1}$ has the maximum error and $error_{i_2}$ has the second maximum error. Note that all of these can be computed in linear time. Based on Lemma 6, we can deduce that $error(Q, P)$ cannot be larger than $error_{up}(Q, P) = error_{i_1} + error_{i_2}$. Our hill-climbing algorithm uses the upper bound, $error_{up}(Q, P)$, to evaluate the effectiveness of a BP-Cube P.

(3) Adjustment. To adjust the current BP-Cube, we try to move a single partition point from one place to another. The heuristic is to move the partition point to either i_1 or i_2. This is because the goal is to reduce $error_{up}(Q, P)$ and moving to i_1 (or i_2) is very likely to reduce $error_{up}(Q, P)$. To decide which partition point should be moved away,

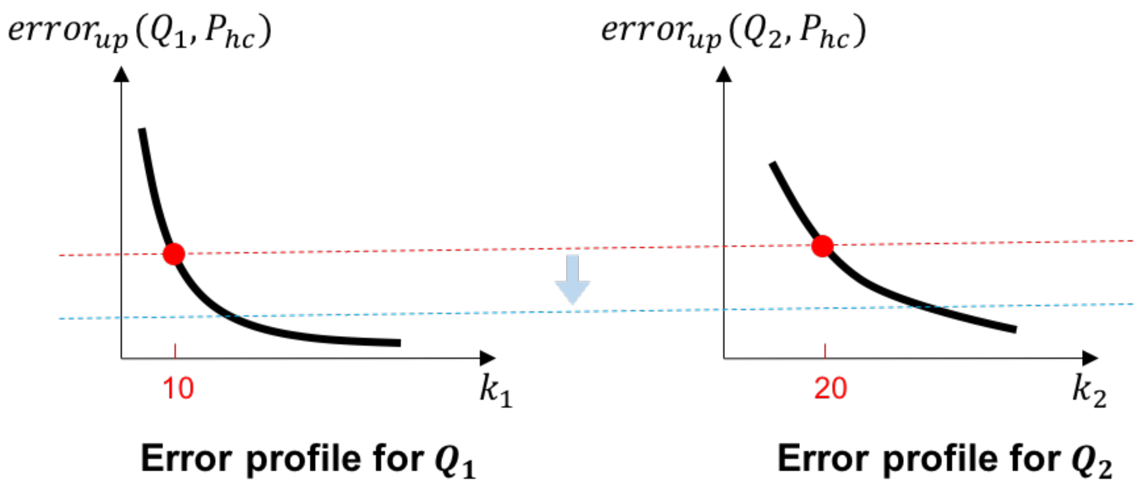


Figure 3.6: An illustration of the binary search algorithm to search for the BP-Cube's shape $k_1 \times k_2$ (suppose $k = 500$).

we want to find the one such that moving it away will have the least *chance* to increase $error_{up}(Q, P)$. Imagine a partition point t is moved away. Only the points between the two partition points nearby t may have a larger $error_i$. Thus, the *chance* that $error_{up}(Q, P)$ will increase depends on the maximum error among the changed points. For example, suppose $P = \{\text{SUM}(1:3), \text{SUM}(1:6), \text{SUM}(1:9), \text{SUM}(1:12)\}$. There are four partition points: 3, 6, 9, and 12. If the partition point $t = 6$ is moved away, only the points in $(3, 9)$ may have a larger $error_i$; for the others, $error_i$ stays unchanged. We compute the maximum $error_i$ among the changed points, i.e., $\max_{i \in (3,9)} error_i$ (after moving 6). Similarly, we compute the maximum error for the other three partition points, i.e., $\max_{i \in (1,6)} error_i$ (after moving 3), $\max_{i \in (6,12)} error_i$ (after moving 9), $\max_{i \in (9,12)}$ (after moving 12). Suppose $\max_{i \in (6,12)} error_i$ is minimal among the four values. Then, the partition point 9 will be moved away.

(4) Stop Condition. The hill-climbing algorithm will stop when $error_{up}(Q, P)$ cannot be decreased through the adjustment process.

Remark. Intuitively, there are two places that could cause the greedy approach to be not optimal. First, the approach aims to optimize the upper bound of the query-template error rather than the query-template error itself. Second, when putting a partition point to a new position, this new position (i.e., i_1 or i_2) is selected heuristically, which may not be the optimal position.

3.6.2 Multidimensional Query Template

Consider a query template, $Q : [\text{SUM}(A), C_1, C_2, \cdots, C_d]$. Given a threshold k, we need to first assign k_i to each dimension C_i such that $\prod_{i=1}^{d} k_i \leq k$. Once $k_1, k_2, \cdots, k_d$ are determined, we apply the above hill climbing algorithm to choosing the k_i partition points in each dimension.

Determine the BP-Cube's Shape. Without loss of generality, we use a 2-dimensional query template Q : $[\text{SUM}(A), C_1, C_2]$ to illustrate our idea. A naive solution is to assign the same value to C_1 and C_2, i.e., $k_1 = k_2 = \sqrt{k}$. But, this ignores the fact that the data distributions in C_1 and C_2 can be quite different. To better balance the values of k_1 and k_2, we first plot an *error profile* for Q_1 : $[\text{SUM}(A), C_1]$ and Q_2 : $[\text{SUM}(A), C_2]$, respectively. The error profile shows how $error_{up}(Q_i, P_{hc})$ decreases with the increase of k_i for $i = 1, 2$, where P_{hc} is the BP-Cube determined by the hill-climbing algorithm. Since it is expensive to compute every data point on the profile curves, we compute a small subset of them and approximate the remaining ones by interpolation. The function used for the interpolation is proportional to $\sim 1/\sqrt{k}$ (see Lemma 4). Once the error profiles are plotted, k_1 and k_2 can be efficiently determined. For example, consider the two error profiles in Figure 3.6. Suppose k = 500. We do a binary search on the y-axis of the error profiles. At each iteration, if $k_1 \cdot k_2 < k$ (or $k_1 \cdot k_2 > k$), it continues the search in the lower (or upper) half of the search range, eliminating the other half from consideration. In the example, suppose the red line is the current search position. Then, we obtain $k_1 = 10$ and $k_2 = 20$ from the error profiles. Since $k = 500$ and $k_1 \times k_2 = 200 < k$, it means that we have the additional budget to enlarge k_1 and k_2 for getting a smaller error. Thus, the next search position is the lower half of the search range (i.e., the blue line). The binary search repeats until $k_1 \times k_2 = k$ or the search range is empty.

Putting It All Together. Given a query template $Q = [\text{SUM}(A), C_1, C_2, \cdots, C_d]$ and a threshold k, the aggregate-precomputation contains two stages. The first stage determines which BP-Cube should be precomputed, and the second stage is to precompute the BP-Cube. Please note that the first stage is based on a sample S. It consists of two steps: (1) Determine the BP-Cube's shape, $k_1 \times k_2 \times \cdots \times k_d$; (2) Run the hill-climbing based algorithm to get k_i partition points from each dimension ($i \in [1, d]$). In the second stage, we need to scan the full data. Ho et al. [91] proposed an efficient algorithm to compute a BP-Cube. Since a BP-Cube is typically several orders of magnitude smaller than the P-Cube, it incurs much less preprocessing cost (see Appendix A.2 for a detailed cost analysis).

3.7 Extensions

Aggregation Functions. As mentioned in Section 3.4.2, AQP++ has an estimator (Equation 3.5) that works for any aggregation function that AQP can support. Each aggregation function, however, may require a different way to construct a precomputed aggregate query set. In this work, we propose an aggregate-precomputation technique for SUM queries. The technique can be extended to COUNT and AVG with small changes. For COUNT queries, we add a virtual attribute to the table with all the values equal to 1. Any COUNT query can be rewritten as a SUM query on the virtual attribute. For AVG queries, since $\text{AVG}(A) = \frac{\text{SUM}(A)}{\text{COUNT}(A)}$, to construct a BP-Cube for it, we need to consider the accuracy of both SUM(A) and COUNT(A). We employ a simple heuristic approach [64] to combine them together, $\alpha \cdot \text{SUM}(A) + (1-\alpha) \cdot \text{COUNT}(A)$, where $\alpha \in [0,1]$ ($\alpha = 0.5$ by default). Given an AVG query template $[\text{AVG}(A), C_1, \cdots, C_d]$, we add a virtual attribute A' to the table with each value equal to $\alpha \cdot A + (1-\alpha)$, and then apply the aggregate-precomputation technique to $[\text{SUM}(A'), C_1, \cdots, C_d]$ to determine the BP-Cube that need to be precomputed.

For holistic aggregation functions such as median and percentile, BP-Cubes cannot support them well. The main reason is that their query answers cannot be easily combined. For instance, it is easy to add the answers to SUM(1 : 5) and SUM(6 : 10) to get the answer to SUM(1 : 10), but such idea will not work for median or percentile. If there is a sophisticated method to handle precomputed percentile results (like BP-Cubes for SUM), one can still benefit from AQP++ by using bootstrap to compute the confidence interval of $q - pre$ and using Equation 3.4 to get the estimation. In the future, we will explore other forms of cubes to handle holistic aggregation functions.

Group-by Queries. We now discuss how to extend AQP++ to support group-by queries. We can treat group-by attributes as condition attributes in the aggregate pre-computation stage and then apply the same hill-climbing algorithm to generate BP-Cubes. For example, suppose a user wants to run queries in the following form:

```
SELECT SUM(sales) FROM table
WHERE age GROUP BY country
```

The user can specify a query template [SUM(sale), age, country], and then use our algorithm to generate a BP-Cube for the template.

In the aggregate identification stage, given a group-by query, e.g.,

```
SELECT SUM(sales) FROM table
WHERE 19<age<31 GROUP BY country
```

we need to identify a precomputed aggregate value for each group. One idea is to apply our aggregate-identification approach to each group one by one. However, this may be costly when the number of groups is large. To make this process more efficient, we can adopt a heuristic approach, where we consider all groups as the same and only apply our approach to the following query (which is obtained by removing the group-by clause from the user query):

```
SELECT SUM(sales) FROM table WHERE 19<age<31.
```

Suppose the identified range condition is "20<age<30". Then, we use it for all groups, and construct the following query

```
SELECT SUM(sales) FROM table
WHERE 20<age<30 GROUP BY country,
```

where we can identify a precomputed aggregate for each group.

For both stages, it is clear to see that the proposed extension may not be the most effective solution to enable AQP++ to support group-by queries. For example, in the aggregate pre-computation stage, it ignores the fact that *country* is a categorical attribute, and its range condition can only use an equal sign, i.e., "*country* = x". We will explore these opportunities and further enhance the extension to group-by queries in future work.

Data Updates. When the underlying data is updated, AQP++ does not only need to update sample data (like AQP), but also needs to maintain precomputed query results. The latter is essentially a materialized view maintenance problem. Many techniques have been proposed to solve the problem [70]. In particular, for SUM, COUNT, AVG queries, their query results can be maintained more efficiently due to the availability of incremental algorithms. Furthermore, since AQP++ only needs to maintain a BP-Cube, it incurs much less maintenance cost than AggPre.

There are many interesting problems in this space, such as how to develop an incremental hill-climbing algorithm, how to achieve a better trade-off between maintenance cost, query response time, and answer quality, and how to combine stale prefix cubes with data updates to answer queries. However, addressing these problems is beyond the scope of this work. We will systematically study these problems in future work and propose a comprehensive solution to data updates.

Multiple Query Templates. The work studies how to decide which BP-Cube should be precomputed for a single query template. When multiple query templates are given, we need to decide how to allocate the space budget to each query template. Suppose there are two query templates Q_1 : [SUM(A_1), C_1, C_2] and Q_2 : [SUM(A_2), C_3], and the space budget is k. A simple approach is to allocate $k/2$ to each one. To better balance the allocation, we can adopt a similar idea with the binary-search algorithm in Section 3.6.2, which tunes the space-budget allocation iteratively. In the beginning, both Q_1 and Q_2 will be allocated to have the space budget of $k/2$. Then, we use error profile curves to estimate which query template has a larger error, e.g., Q_1 has a larger error. In this situation, Q_1 needs more space budget. We adjust the allocation by assigning the budget of $3k/4$ to Q_1 and the budget of $1/k$ to Q_1. The iterative process will continue until the search range is empty.

Space Allocation. In AQP++, part of the space is used for sampling while the other part is used for storing BP-Cubes. One natural question is how to allocate this budget between the two to achieve the best performance given a fixed budget. We adopted a simple approach in this work. This approach is based on the observation that sample size significantly impacts query response time, but the size of BP-Cubes does not. Therefore, we can first select the maximum sample size that meets the user's requirement for response time (e.g., less than 0.5 s) and then use the remaining space for storing BP-Cubes. In the future, we will study how to leverage query workloads to develop a more sophisticated approach.

3.8 Experimental Results

We conduct extensive experiments to evaluate AQP++. The experiments aim to answer three major questions. (1) When does AQP++ give more accurate answers than AQP? (2) How does AQP++ compare with AQP and AggPre in terms of preprocessing cost, response time, and answer quality? (3) How well does the hill climbing based approach perform compared to the equal-partition scheme?

3.8.1 Experiment Setup

Experimental Settings. We implemented AQP and AQP++ using DBX, a commercial OLAP system with column-store indexes supported. The code was written in C++, compiled using Visual Studio 2015, and connected to DBX through ODBC. The experiments were run on a Windows machine with an Intel Core 8 i7-6700 3.40GHz processor, 16GB of RAM, and 1TB HDD.

Datasets. We conducted experiments on three datasets. (1) *TPCD-Skew* is a synthetic dataset generated from the TPCD-Skew benchmark [67]. We generated 100GB data with the skew parameter $z = 2$ and ran queries on the lineitem table, which contains 600 million rows. (2) *BigBench* is a synthetic dataset generated from the Big Data Benchmark [1]. We generated 100GB of data and ran queries on the UserVisits table, which contains 752 million rows. (3) *TLCTrip* is a real-world dataset from the NYC Taxi and Limousine Commission [3]. We used the yellow car data from 2009 to 2016, which is of size 200GB and contains 1400 million rows.

Sampling. It is worth noting that AQP++ is a general framework that can connect any AQP engine with AggPre no matter which sampling approach the AQP engine adopts. To allow for a clear comparison between the cores of AQP and AQP++ frameworks (i.e., Equation 3.3 vs. Equation 3.4), we assume that both AQP and AQP++ only use a uniform sample by default. Specifically, we create a uniform sample from the full table and store the sample in DBX as a table (sample rate = 0.05% by default). To answer queries, the sample will be used by AQP and AQP++. To evaluate the performance of AQP++ on other forms of samples, we implemented another two sampling approaches, measure-biased sampling [78] and stratified sampling [47], used by the state-of-the-art AQP systems, and compared AQP with AQP++ on these samples.

Error Metrics. We adopted *relative error* to quantify query accuracy because it is easy to interpret. For an approximate query result $\hat{q} \pm \epsilon$, where ϵ is half the width of the 95% confidence interval, the relative error of the query is defined as $\frac{\epsilon}{q}$, where q is the true answer of the query. Given a collection of queries, when we say median (or average) error, it refers to the median (or average) value of the relative errors of the queries in the collection.

Table 3.1: Comparison of the overall performance (TPCD-Skew 100GB, k=50000, 0.05% uniform sample).

	Preprocessing Cost		**Response**	**Answer Quality**	
	Space	**Time**	**Time**	**Avg Err.**	**Mdn Err.**
AQP	51.2 MB	4.3 min	0.60 sec	2.67%	2.48%
AggPre	> 10 TB	> 1 day	< 0.01 sec	0.00%	0.00%
AQP++	51.9 MB	11.7 min	0.67 sec	0.27%	0.19%

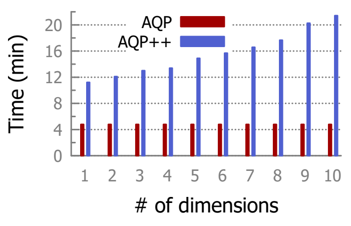
3.8.2 Overall Performance

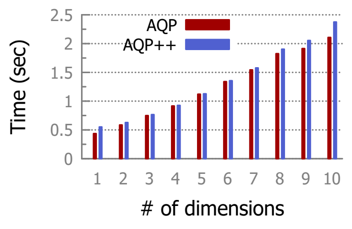
Table 3.1 compares the overall performance (preprocessing cost, query response time, and answer quality) of AQP++ with alternatives on the TPCD-Skew dataset. We randomly generated 1000 queries using the template of [SUM(l_extendedprice), l_orderkey, l_suppkey], where the query selectivity is between $0.5\% - 5\%$.

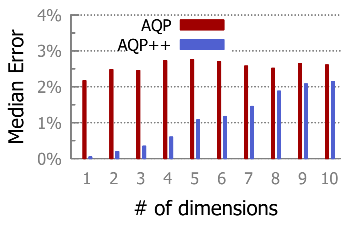
Suppose the latency requirement is 1 second. We first examine whether the state-of-the-art OLAP solutions can meet the requirement. We first chose a commercial (M)OLAP system and created a data cube with l_extendedprice as the measure attribute and l_orderkey and l_suppkey as the dimension attributes. The cube has a hierarchy structure of "l_suppkey →l_orderkey". We ran the 1000 queries over the cube and found that the cube size was around 4GB and the average query response time was more than 10 seconds, which is far from interactive. The reason is that the two dimensions, $\langle l\_orderkey, l\_suppkey\rangle$, have a large number of distinct values (i.e., 377 million), thus a range query still needs to scan a lot of cells in the cube. In addition, we tested the time of directly executing queries in DBX. DBX needed an average response time of 6 seconds and a maximum response time of 35 seconds to run all the queries, which did not meet the latency requirement either.

We now evaluate the performance of AQP, AggPre, and AQP++. AQP used a uniform sample to answer queries (sample rate = 0.05%); AggPre precomputed the complete P-Cube using the algorithm in [91]. In the lineitem table, l_orderkey and l_suppkey have 1.5×10^8 and 7.5×10^4 distinct values, respectively, so there are 1.1×10^{13} cells in P-Cube. Clearly, AQP and AggPre represented two extreme cases of AQP++, where one did not precompute any aggregate, and the other precomputed all possible aggregates. For AQP++, we used the same sample as AQP and precomputed a BP-Cube of size $k = 50,000$.

Table 3.1 compared its performance with AQP and AggPre. We can see that all of them met the latency requirement (< 1 sec), but they were quite different in terms of preprocessing cost and answer quality. AQP++ spent orders of magnitude less preprocessing time and space than AggPre since it only needs to precompute a small BP-Cube rather than the complete P-Cube. In comparison with AQP, AQP++ reduced the average error by 10× and the median error by 13× for almost the same preprocessing space and about 7.4 minutes more preprocessing time. Furthermore, the overhead added to the AQP++'s response time (due to the aggregate-identification step) is negligible. This is because the response time







(a) Preprocessing Time (b) Response Time

(c) Answer Quality

Figure 3.7: Comparison of AQP and AQP++ by varying # of dimensions (TPCD-Skew 100GB, k=50000, 0.05% uniform sample).

was dominated by the I/O time for reading data from the sample table, and the added CPU time was relatively very small.

To further compare AQP and AQP++, we set the sample rate of AQP to 4% such that it can reach approximately the same average error as AQP++. We call it AQP(large). Compared to AQP++, the AQP(large)'s sample size was about 80× larger, which significantly increased the preprocessing time and space. Furthermore, due to the sample size increase, its query response time was more than 1 second, which violated the latency requirement.

We implemented APA+ [97] and compared its performance with AQP++. Since our query template was 2-dimensional, we assumed that *1-dimensional facts* (i.e., a set of statistics defined by APA+) are available in the system for each query. To process a query, APA+ first gets the related facts and then combines them with a sample to estimate the answer to the query. We used the gurobi library to solve the quadratic programming problem in APA+ such that it can minimize the estimation error. The experimental result showed that APA+ achieved a median error of 1.69% while the median error for AQP++ was only 0.19%. The reason is that APA+ does not use BP-Cubes for result estimation, while AQP++ can identify the best BP-Cube to precompute and use it for result estimation.

3.8.3 Detailed Performance

In this section, we evaluate the performance of AQP++ by varying the number of dimensions and the set of condition attributes, aiming to gain a deeper understanding of various trade-offs. We also examine the effectiveness of the hill climbing algorithm for aggregate precomputation. If not specified, we use the same dataset and queries as the previous section and set the sample rate to 0.05%, k=50000, and the number of dimensions to 2 by default.

Number of Dimensions. We compare the preprocessing time, response time, and answer quality of AQP and AQP++ by varying the number of dimensions. We chose ten columns from the lineitem table and constructed ten query templates accordingly:

[SUM (l_extendedprice), l_orderkey],

[SUM (l_extendedprice), l_orderkey, l_partkey],

...

[SUM (l_extendedprice), l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_discount, l_tax, l_shipdate, l_commitdate, l_receiptdate],
where each of them has a different number of dimensions. We compared AQP with AQP++ w.r.t. each query template, and reported the result in Figure 3.7.

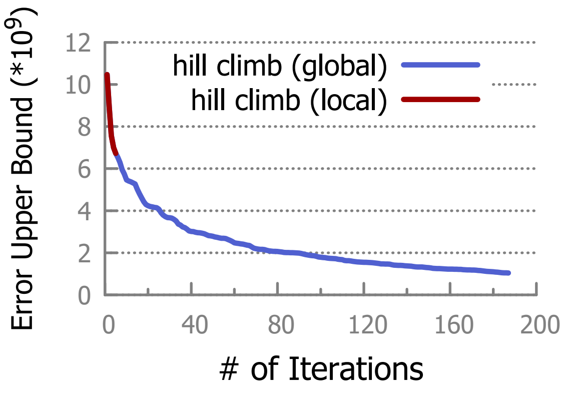
Figure 3.7(a) compares the preprocessing time of AQP and AQP++. Since AQP only needs to create a random sample, the number of dimensions had no impact on its preprocessing time. In comparison, AQP++ requires a little more preprocessing time when the number of dimensions increases since it needs to generate an error profile for each dimension. The larger the number of dimensions, the more time is spent generating error profiles.

Figure 3.7(b) shows how the response time changed w.r.t. the number of dimensions. To identify the best-precomputed aggregate value, AQP++ first generates $4^d + 1$ candidate values and then uses a subsample to estimate which will lead to the smallest error. As the number of dimensions increased, the number of candidate values increased exponentially. However, we can see from the figure that the difference between the response times of AQP and AQP++ did not increase exponentially. This is because if the number of candidate values is increased by four times, AQP++ will decrease the subsampling rate by four times as well, which helps to reduce the aggregate-identification time.

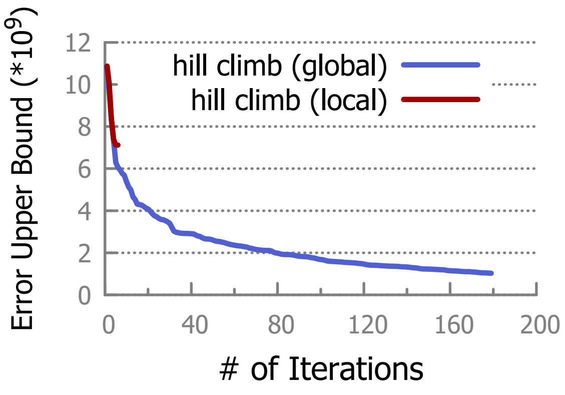
Figure 3.7(c) compares the answer quality between AQP and AQP++ in terms of median error. The figure shows that the median error of AQP++ got bigger as the number of dimensions increased. This is because that the space budget $k = 50000$ was fixed. If there is only one dimension, this dimension can be assigned a budget of 50000 partition points, but if there are two dimensions, each dimension (on average) can only be assigned a budget of $\sqrt{50000} = 224$ partition points, which is less effective than 1D. Nevertheless, as shown in Figure 3.7(c), AQP++ outperformed AQP by 12.8× for 2D. As the number of dimensions increased, the improvement of AQP++ over AQP decreased. The result indicates that AQP++ can scale up to 10 dimensions but is hard to scale to a very large number of dimensions (e.g., 20) due to the limitation of prefix cubes.

Hill Climbing. We evaluate the adjustment approach of our hill climbing algorithm. Recall that our approach considers all partition points at each iteration and picks the best one to move. One may ask why not only consider the four partition points next to i_1 and i_2. We compared the two adjustment approaches.

As discussed in Section 3.6, an equal partitioning scheme is ineffective when attributes are highly correlated. Thus, we picked up two attributes, l_shipdate and l_commitdate, that strongly correlate with l_extendedprice, and constructed the query template: [SUM (l_extendedprice), l_shipdate, l_commitdate]. Figure 3.8 compares the upper bound of the query template error of Hill Climb (global) and Hill Climb (local) on each dimension, where the former used our adjustment approach while the latter adopted the alternative. We set



(a) l_shipdate



(b) l_commitdate

Figure 3.8: Evaluation of adjustment approach of hill climbing (TPCD-Skew 100GB, $k_1 = 200$, $k_2 = 200$, and 0.05% sample).

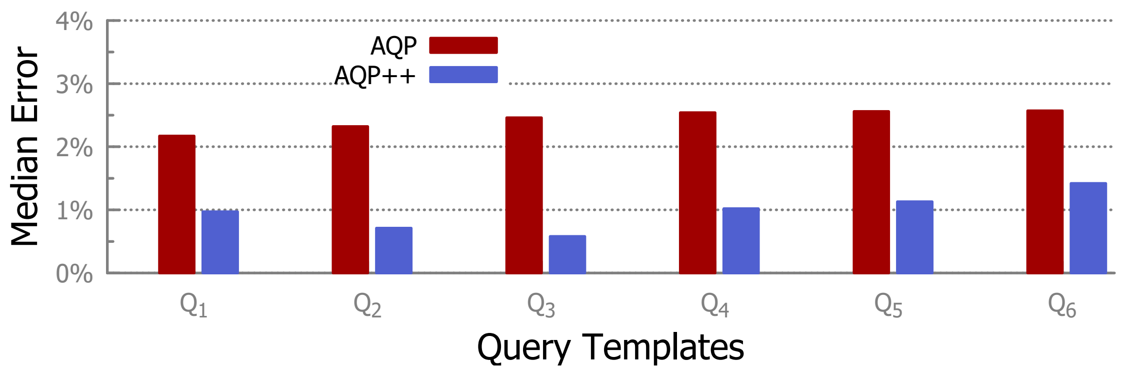


Figure 3.9: Evaluation of the changes of the set of condition attributes in user queries. Note that only Q_3 has a precomputed BP-Cube (TPCD-Skew 100GB, k=50000, 0.05% sample).

$k_1 = k_2 = 200$. We can see that Hill Climb (local) converged to a local optimum with less than ten iterations while Hill Climb (global) can continue the iterative process and finally reach a much better result. The reason is that Hill Climb (local) only considers the four partition points next to i_1 and i_2. The algorithm will stop if moving them away cannot lead to a better solution.

Changes of Condition Attributes. In exploratory workloads, a user may frequently change the set of condition attributes in her queries. We discuss how AQP++ handles this situation below.

Suppose a user may issue a collection of queries generated from the three query templates: $Q_1 : [\texttt{SUM}(A), C_1]$, $Q_2 : [\texttt{SUM}(A), C_1, C_2]$, $Q_3 : [\texttt{SUM}(A), C_1, C_2, C_3]$, but only Q_2 has a precomputed BP-Cube P_2. We next show how AQP++ can use P_2 to answer the queries from Q_1 and Q_3.

If a user query q is from Q_1, we can rewrite q as an equivalent query q' from Q_2 where q' does not enforce any restriction on C_2, thus AQP++ can still use P_2 to answer q. For example, consider $q : [\texttt{SUM}(A), 1:2]$. It can be rewritten as $q' : [\texttt{SUM}(A), 1:2, 1:|\mathsf{dom}(C_2)|]$ (where C_2 can be any value in its domain), and then be processed by AQP++ using P_2.

If a user query q is from Q_3, we can consider P_2 as a 3-dimensional BP-Cube P_2'. For example, suppose the shape of P_2 is $k_1 \times k_2$. Then, it can be seen as a 3-dimensional BP-Cube P_2' with the shape of $k_1 \times k_2 \times 1$. Thus, AQP++ can still use P_2 to answer q.

To evaluate this approach, we constructed six query templates $Q_1 : [\texttt{SUM}(A), C_1], \cdots, Q_6 : [\texttt{SUM}(A), C_1, C_2, C_3, C_4, C_5, C_6]$ on the TPCD-Skew dataset, where A is l_extendedprice, and $C_1, C_2, \cdots, C_6$ are l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_discount, respectively. We assumed that only Q_3 had a precomputed BP-Cube (with size $k = 50000$). We randomly generated 1000 queries from each query template with the selectivity of 0.5%-5%. Figure 3.9 compares the median error of AQP and AQP++ for these queries w.r.t. each Q_i ($i \in [1, 6]$). We can see that AQP++ kept outperforming AQP when changing the set of condition attributes from Q_3 to Q_1 or from Q_3 to Q_6, but with the improvement being smaller as more changes are made. An interesting future work is to study how to detect this situation and how to trigger the computation of more suitable BP-Cubes in an automatic way.

3.8.4 Evaluation With Other Sampling Methods

The previous experiments validate the effectiveness of AQP++ on uniform samples. In this section, we implement two other sampling approaches used by the state-of-the-art AQP systems [78, 47] and examine the performance of AQP++ on these samples.

AQP (measure-biased) vs. AQP++ (measure-biased). Measure-biased sampling selects each record with a probability proportional to the value in the measure attribute. That is, the larger the value in the measure attribute, the more likely the record will appear in the sample. This has shown be to a very effective sampling approach to mitigate the negative impact of outliers on estimated answers. We randomly generated 1000 queries with the selectivity of $0.5\% - 5\%$ using the default template. Since measure-biased sampling is designed for handling outliers, we only chose the queries that can cover (at least) one outlier, where a value is defined as an outlier if l_extendedprice is larger than $\mathsf{median}(\mathsf{l\_extendedprice}) + 3 * \mathsf{SD}(\mathsf{l\_extendedprice})$. We created a 0.05% measure-biased sample of the dataset, leading to a sample of size $|S| = 0.3$ million, and then compared AQP with AQP++ on the sample by varying BP-Cube size from k=1000 to k=10,000. Figure 3.10(a) plots the median error. We can see that with a very small BP-Cube (e.g., k=5000), AQP++ reduced the median error of AQP by 3.3×, which validated the effectiveness of AQP++ for measure-biased sampling.

AQP (stratified) vs. AQP++ (stratified). Stratified sampling divides data into different groups and then applies a different sampling ratio to each group. The sampling ratio of each group is disproportional to its group size. This is to ensure that there are enough records being sampled from small groups. Since stratified sampling is designed for opti-

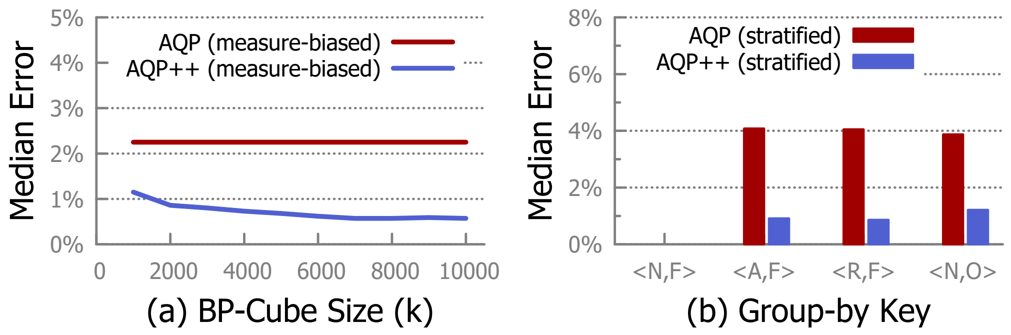


Figure 3.10: Comparing AQP++ with AQP using measure-based sampling and stratified sampling (TPCD-Skew 100GB).

mizing group-by queries, we randomly generated 1000 group-by queries of the following form:

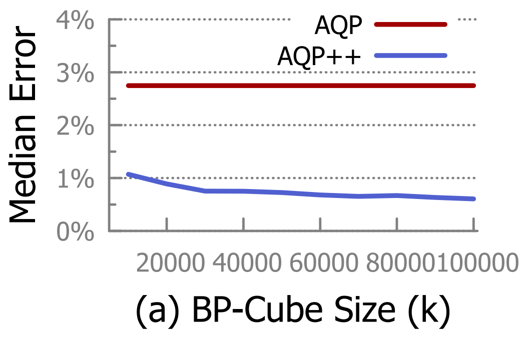
```
SELECT SUM(l_extendedprice) FROM lineitem
WHERE  l_orderkey, l_suppkey
GROUP BY l_returnflag, l_linestatus,
```

where the selectivity of each query is between 0.5% - 5%. We then created a 0.05% stratified sample of the dataset w.r.t. the group-by attributes, l_returnflag and l_linestatus, leading to a sample of size $|S| = 0.3$ million. AQP used the sample to estimate the answers to the group-by queries; AQP++ used the same sample along with a small BP-Cube of size $k = 50,000$ to estimate the answers. Figure 3.10(b) reports the median error w.r.t. each group. We can see that AQP++ achieved $3\times -4\times$ more accurate answers than AQP, which validated the effectiveness of AQP++ for stratified sampling. Interestingly, both AQP++ and AQP returned true answers for the group-by key of "<N,F>" because the group size was very small and all its records were included into the sample due to the use of stratified sampling.

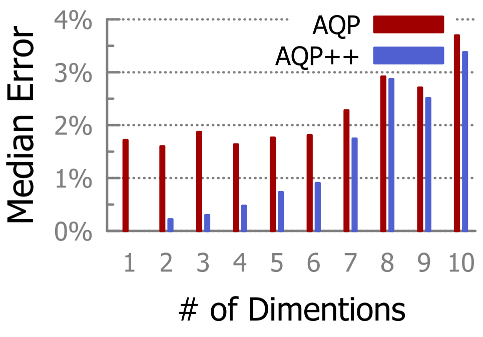
3.8.5 Evaluation on More Datasets

We compared the performance of AQP++ and AQP on two other datasets, BigBench and TLCTrip.

For the BigBench dataset, we want to examine the performance of AQP++ for different BP-Cube size. We created a 0.05% uniform sample of the dataset, and randomly generated 1000 queries using the template of [SUM (adRevenue), visitDate, duration, sourceIP] with the selectivity of 0.5%-5%. Figure 3.11(a) compares the median error of AQP++ and AQP by varying k. We can see that even with a small BP-Cube, AQP++ can still outperform AQP by a lot. For example, when $k = 50,000$, AQP++ reduced the median error by $3.8\times$. As k grows, AQP++ can achieve better and better performance, finally reached a median error of 0.60% when $k = 100,000$.



(a) BigBench



(b) TLCTrip

Figure 3.11: Comparing AQP++ with AQP on the BigBench (100 GB) and TLCTrip (200GB) datasets.

For the TLCTrip dataset, we want to examine the performance of AQP++ for different number of dimensions. We chose ten columns from the table, and constructed ten query templates accordingly: [SUM (Distance), Pickup_Date], $\cdots$, [SUM (Distance),Pickup_Date, Pickup_Time,vendor_name,Fare_Amt,Rate_Code, Passenger_Count, Dropoff_Date, Dropoff_Time, surcharge, Tip_Amt]. We created a 0.1% uniform sample of the dataset; for each query template, we randomly generated 1000 queries with the selectivity of 0.5%-5%, and precomputed a BP-Cube of size $k = 300,000$ for it. Figure 3.11(b) compares the median error of AQP++ and AQP w.r.t. each query template. Similar to Figure 3.7(c), we found that AQP++ significantly outperformed AQP when the number of dimensions is small and marginally improved the median error of AQP when the number of dimensions was increased to 10.

3.9 Conclusion

In this work, we studied how to enable database systems to answer aggregation queries within interactive response times. We found that the two separate ideas for interactive analytics, AQP and AggPre, can be connected together using the AQP++ framework. We presented the unification and generality of the framework, and demonstrated (analytically and empirically) why AQP++ can return a more accurate answer than AQP. After that, an in-depth study of the framework was conducted for range queries. In the study, we formally defined the aggregate-identification and aggregate-precomputation problems, and proposed both optimal solutions (under certain assumptions) as well as effective heuristic approaches (for general settings). We implemented AQP++ on a commercial OLAP system, and evaluated them on three datasets. Experimental results showed that AQP++ can improve the answer quality of AQP by up to 10× and reduce the preprocessing cost (both time and space) of AggPre by several orders of magnitude.

Our work is a first attempt to provide a general framework to connect AQP and AggPre together. Since both AQP and AggPre have been extensively studied in the past, we believe there are many future research directions to explore. First, various techniques have been proposed to optimize AQP (e.g., workload-driven sample creation) as well as AggPre (e.g., cube approximation). It would be interesting to revisit these techniques under the AQP++ framework. Second, there are some aggregation functions that AQP cannot handle well, such as min and max. However, they are easy for AggPre. Since AQP++ connects AQP with AggPre, it would be interesting to explore whether AQP++ can be extended to support these aggregation functions. Third, it might be hard for some users to decide which query templates should be specified. Thus, user-guided query template design is another interesting topic to explore.

Chapter 4

SamComb: Combining Different Types of Samplers

In Chapter 3, we have presented AQP++. It combines samples with cubes (that store precomputed aggregations). One issue with AQP++ is that it suffers from the limitation of cubes. More specifically, cubes only support range queries, and cubes still suffer from the curse of dimensionality. On the other hand, we observed that the sample is a general data summary. It supports more types of queries and does not suffer from the curse of dimensionality. Then, one question is can we improve the estimation quality by combining different types of samples?

In this chapter, we present SamComb, a bandit-based sampler combination framework that combines different types of samplers. Given a set of samplers, a budget, and a query, SamComb can automatically decide how much budget should be allocated to each sampler so that the combined estimation achieves the highest accuracy.

4.1 Motivation

The quality of sample-based estimations highly dependent on two important factors simultaneously (which will be elaborated in Section 4.2.1): i) how data is sampled, i.e., a *sampling distribution* which specifies how likely a tuple is drawn into the sample, and ii) the underlying data distribution in the population where we want to calculate parameters. **One size does not fit all.** Various samplers are proposed in the literature [103, 102, 78, 50] with different sampling distributions. For example, a uniform sampler draws a sample from the population where each tuple has the same probability to be selected [90], a stratified sampler draws a sample so that the tuples in each group have the same probability to appear in the sample [50, 103], and a measure-biased sampler draws a sample so that the tuples with large measure values have a higher probability to be selected [78]. While the sampling distribution has to be specified before the sample is drawn, ii) is decided at the "estimation time".

```
SELECT SUM(A) FROM table WHERE B > 10
```

For example, the above query specifies that we want to focus on the population with `B > 10` and estimate `SUM(A)`. Thus, the distribution of `A` in that particular population and how well it matches the sampling distribution decide the estimation accuracy. Therefore, there is no single *sampler* that works well in all cases.

In this work, we investigate how to combine multiple samplers of the same population to estimate population parameters. Given k different samplers, a *sample budget*, an *aggregation function* (i.e., `COUNT`, `SUM`, `AVG`), and a *predicate* specifying the population to be focused on (e.g., `B > 10`), we study how to allocate the budget to each sampler, so that the combined estimation using k samplers achieves the highest accuracy. For ease of presentation, we use SQL queries (e.g., in the example above) to represent population parameters to be estimated. Similar to existing work [47, 78], we assume each sampler precomputes a large sample, thus they can efficiently draw a sample with given budget by sequentially scanning the data.

Exploitation vs Exploration. We prove that the optimal budget allocation strategy is to allocate all the budget to the *best* sampler. The next questions is how to identify the best sampler. One may design some heuristic rules by leveraging the query information. Unfortunately, it is insufficient to identify the best sampler by only looking into the query, due the following two reasons.

Firstly, a query could satisfy multiples rules, and resulting in multiple samplers chosen by different rules. For example, consider the following two rules: 1) Rule 1: If a query contains `A` in the aggregate function, then choose the measure-biased sampler over `A` column. 2) Rule 2: If a query contains `B = b` in the predicate, where `b` is a constant, then choose a stratified sampler over `B` column. Consider a query:

```
SELECT SUM(price * (1 - tax)) WHERE country = 'USA'
```

The measure-biased sampler over `price` and `tax` column, and the stratified sampler over `country` both satisfy the rules. Hence it is still unknown which sampler should be chosen.

Secondly, even we have a rule-based approach that returns a single sampler, the best sampler still can be different for different data. For example, consider a query:

```
SELECT SUM(price) FROM table WHERE itemid < 100
```

If all tuples satisfy `itemid < 100`, then the measure-biased sampler over `price` column is known as the best sampler. However, if all the tuples whose `itemid < 100` have a very small price, then the measure-biased sampler could perform worse than a uniform sampler, since the tuples that contributes to the answer most are those with small price, and are assigned with a small probability by the sampler.

The above two reasons suggests that it is unknown which sampler is the best w.r.t. a given query before scanning the whole data. To address this issue, we allocate a certain amount of sampling budget to each sampler (i.e., draw samples of certain size using each sampler) in an adaptive way, and assess which sampler is the best using their samples. The identified one is called the *empirically best sampler*, which may or may not be the *truly best sampler*. During this process, we need to balance the *exploitation vs exploration* trade-off, that is, to exploit the the empirically best sampler or explore a different sampler which could be the truly best sampler.

SamComb Framework. To solve this challenge, we propose SamComb, a bandit-based sampler combination framework. It models sampler combination problem as multi-armed bandit (MAB) problem [49, 156, 117], which is a principled way to balance the *exploitation vs exploration* trade-off. SamComb is an iterative framework. At each iteration, it assesses which sampler is the best, then uses an MAB-based approach to decide which sampler to select, and finally allocates a small budget to the sampler. The iteration will continue until the budget is exhausted. Finally, SamComb uses the budget allocated to each sampler to estimate the answer to the query, respectively, and computes a weighted average of these estimated answers as the final answer. Comparing to approaches that select one best sample, the select-then-combine framework of SamComb can fully leverage all the samples and does not waste any budget used for exploration.

The approaches developed for MAB have a guarantee that most budget will be allocated to the best arm. However, it is not clear whether they are still applicable to our problem, since the two problems have different optimization objectives. MAB aims to maximize a linear sum of the reward from each arm and it only involves sample size (i.e., pull times) as variable, while our problem aims to minimize a complex function of the variance from each sampler: it is not linearly additive as MAB, and it involves both sample size and weight. We propose effective solutions and justify our solutions both analytically and empirically.

Firstly, we discuss how to model our problem as an MAB problem and then justify this modeling by showing that their objective functions share some common properties.

Secondly, to solve our budget allocation problem, we extend two well-known MAB approaches [49], ϵ_t-greedy and Upper Confidence Bound (UCB). We prove that both approaches can guarantee that the allocated budget to each sub-optimal sampler is at most $O(\ln n)$, where n is the total budget.

Thirdly, we discuss how to combine the estimation from each sampler into the final estimation. We propose two approaches and prove the optimality of both approaches.

Finally, we conduct extensive experiments to evaluate SamComb using both synthetic and real-world datasets. The results validate the effectiveness of our approaches empirically. We also apply SamComb to applications like selectivity estimation, and demonstrate the advantages of combining multiple samplers. We see this work as an initial step towards building a principled federated framework to address the one-size-does-not-fit-all challenge in online aggregation, approximate query processing, and cardinality estimation.

In summary, we make the following contributions:

Full Table

Id	Price	Country
1	150	USA
2	10	CA
3	0.1	USA
4	350	USA
5	50	CA
6	3	USA
7	650	CA
8	0.5	USA
9	900	USA
10	85	USA
…	…	…
1M	0.2	CA

A sample drawn by uniform sampler

Id	Price	Country	Probability
1	150	USA	1e-6
3	0.1	USA	1e-6
…	…	…	…

A sample drawn by measured-biased

Id	Price	Country	Probability
9	900	USA	0.018
7	650	CA	0.013
…	…	…	…

A sample drawn by stratified sampler

Id	Price	Country	Probability
2	10	CA	2.5e-6
4	350	USA	6.25e-7
…	…	…	…

Figure 4.1: An illustration of different samplers.

1. We propose a novel idea that combines different samplers to enhance sample-based estimation. We formally define the sampler combination problem. To the best of our knowledge, we are the first to study this problem.

2. We model our problem as a multi-armed bandit problem, and propose SamComb, a bandit-based sampler combination framework to solve the problem.

3. We propose two budget allocation approaches based on existing MAB strategies, and two weight allocation approaches to combine the estimated answers from multiple samplers into the final answer. We prove theoretical guarantees for these approaches.

4. We evaluate SamComb on both synthetic and real-world datasets. The results show that SamComb can effectively combine multiple samplers and achieve a higher estimation accuracy compared to baselines.

4.2 Problem Formalization

In this section, we first define the concept of sampler and then formalize the sampler combination problem.

We focus on the population parameters that can be expressed in the following form of SQL queries:

```
SELECT f(A) FROM table
WHERE Predicate (B1, B2, ...)
```

where f is `SUM`, `COUNT`, or `AVG`. For ease of presentation, we mainly consider $f =$ `SUM`, since `COUNT` is a special case of SUM and `AVG` is the ratio of `SUM` to `COUNT`. We also discuss how to support other aggregate functions such as `PERCENTILE` in Section 4.6.

4.2.1 Sampler

There are different sampling methods (uniform, measure-biased, and stratified sampling). We find that each of them can be seen as a special case of weighted sampling. Thus, we define sampler as weighted sampling. Given a table T with N tuples, a sampler draws a weighted sample from T such that the probability of each tuple being selected is proportional to its weight. Definition 4 presents a formal definition.

Definition 4 (Sampler)**.** *Given a table T with N tuples, each tuple $t_i \in T$ is associated with a probability p_i, where $0 < p_i \leq 1$ and $\sum_{i=1}^{N} p_i = 1$. A sampler $\mathbb{S}$ can draw a weighted sample S with replacement of any given sample size from T, where each tuple $t_i \in T$ has a probability of p_i to be sampled.*

If the context is clear, we represent a sampler by its probability of selecting each tuple, i.e., $\{p_1, p_2, \cdots, p_N\}$. The only difference among different sampling methods is how to compute p_i (for all $i \in [1, N]$). Note that one can certainly obtain other samplers by computing p_i differently. Our system supports them as well.

- *Uniform Sampler:* A uniform sampler selects each tuple with the same probability, i.e., $p_i = \frac{1}{N}$.
- *Measure-biased Sampler:* Given a measure column, a measure-biased sampler selects each tuple with probability proportional to the measure value, i.e., $p_i = \frac{m_i}{\sum_{i=1}^{N} m_i}$, where m_i is the measure value for tuple i.
- *Stratified Sampler:* Given a stratified column G, the table is divided into $|\mathsf{dom}(G)|$ groups, where $|\mathsf{dom}(G)|$ is the domain size of G (i.e., the number of distinct values in G). A stratified sampler selects each group with equal probability. For each tuple i, let G_i be the group that contains tuple i, then we have $p_i = \frac{1}{|\mathsf{dom}(G)||G_i|}$.

Example 5 illustrates how each sampler computes p_i.

Example 5. *Consider the example in Figure 4.1. The full table has 1M tuples. A uniform sampler selects each tuple with equal probability $\frac{1}{1M}$ = 1e-6. Now, consider a measure-biased* *sampler on Price column. Suppose the total measure of price is* $150 + 10 + ... + 0.2 = 50000$*, then the probability of selecting tuple t is computed as* $\frac{t.price}{50000}$*. E.g., the probability of selecting tuple* 9 *is* $\frac{900}{50000} = 0.018$*. Finally we consider a stratified sampler on Country column. It has 2 strata: USA and CA, where USA has* $0.8M$ *tuples and CA has* $0.2M$ *tuples. Hence, for the tuples whose country is USA we have* $p_i = \frac{1}{2*0.8M}$ *= 6.25e-7, and for the tuples whose country is CA we have* $p_i = \frac{1}{2*0.2M}$ *= 2.5e-6.*

Answer Estimation. Given a query q and a weighted sample S, let $q(S)$ denote the estimated answer based on S. Hansen-Hurwitz (HH) estimator [88] can be directly applied to estimate the answer to an SUM query when the query has no predicate (e.g., `q:SELECT SUM(Price) FROM table`). See Equation (4.1) below.

$$q(S) = \frac{1}{|S|} \sum_{i=1}^{|S|} \frac{y_i}{p_i}, \tag{4.1}$$

where y_i is the aggregate value (e.g., $t_i[Price]$) of tuple i.

The HH estimator can be easily extended to support predicates through query rewriting. The main idea is to rewrite a with-predicate query as an equivalent without-predicate query. E.g., query `SELECT SUM(Price) From table WHERE Predicate` can be rewritten as: `SELECT SUM(CASE WHEN Predicate THEN Price ELSE 0 END) FROM table`. Then, we generalize the $\frac{y_i}{p_i}$ to d_i: $d_i = 0$ if the tuple does not satisfy the predicate, otherwise $d_i = \frac{y_i}{p_i}$.

Estimation Quality. A common approach to measure the estimation quality are confidence intervals, which bound the real result with high probability. To compute the confidence interval, we first define *sampler quality.*

Definition 5 (Sampler Quality)**.** *Given a sampler* $\{p_1, p_2, \cdots, p_N\}$ *and a SUM query, we define a discrete distribution* $\mathcal{D}^q$ *w.r.t. query q (abbreviated as* $\mathcal{D}$ *if the context is clear) that takes value* d_i *with probability* p_i *for each* $i \in [1, N]$*. Define sampler quality by* $\mathsf{var}(\mathcal{D})$*, which is computed as*

$$\mathsf{var}(\mathcal{D}) = \sum_{i=1}^{N} p_i \cdot \Big(d_i - \mathsf{mean}(\mathcal{D})\Big)^2, \tag{4.2}$$

where $\mathsf{mean}(\mathcal{D}) = \sum_{i=1}^{N} p_i \cdot d_i$*.*

Intuitively, the sampler quality measures how good a sampler is for answering a given query. It is the variance of the HH estimator with a sample of 1 tuple drawn from the sampler. Hence, after drawing the same size of samples, the sampler with a higher quality (lower variance) will result in a better estimator.

As each tuple is drawn independently, the variance of the estimator over a sample of size S can be expressed as $\frac{\mathsf{var}(\mathcal{D})}{|S|}$. Based on the central limit theorem (CLT), the confidence interval is computed as

$$CI = q(S) \pm \lambda\sqrt{\frac{\mathsf{var}(\mathcal{D})}{|S|}} \tag{4.3}$$

where λ is a constant number related to the confidence level. For example, $\lambda = 1.96$ means the true value lies in the confidence interval with the probability of 95%.

The larger the width of the confidence interval (i.e., $\lambda\sqrt{\frac{\mathsf{var}(\mathcal{D})}{|S|}}$), the lower quality the estimated answer. Given a query, we prefer a sampler with $\mathsf{var}(\mathcal{D})$ as small as possible. Thus, $\mathsf{var}(\mathcal{D})$ is named *sampler quality* in Definition 5.

For simplicity, let $\mathsf{var}(q(S)) = \frac{\mathsf{var}(\mathcal{D})}{|S|}$ be the variance of $q(S)$. Then, the confidence interval can also be denoted by

$$CI = q(S) \pm \lambda\sqrt{\mathsf{var}(q(S))} \tag{4.4}$$

Sampler Implementation. Each sampler exposes an interface that takes a sample budget n_i as input and returns a sample of size n_i. To save sampling time, we assume that each sampler has precomputed a large sample stored on disk in the offline stage. This is achieved by applying the approach in Appendix B of Sample + Seek [78]. During query time, it sequentially scans n_i tuples from the precomputed sample to get a sample of size n_i, or directly computes the aggregate on a sample by issuing a range query with predicate `row_id BETWEEN a AND b`.

4.2.2 Sampler Combination Problem

Problem Definition. Suppose the total sample budget is n. The sampler combination problem is to study how to allocate the total budget to each sampler such that the *combined estimator* performs the best. Specifically, given a query q, we draw a sample S_i with size n_i from sampler $\mathbb{S}_i$, such that $\sum_{i=1}^{k} n_i = n$. For each sample S_i, we can get an unbiased estimator of the query result, i.e., $q(S_i)$. Then we combine k estimators into the final estimator. Let ψ be the set of drawn samples, i.e., $\psi = \{S_1, S_2, ..., S_k\}$. We denote their combined estimator as $q(\psi)$, which is computed as follows:

$$q(\psi) = \sum_{i=1}^{k} w_i \cdot q(S_i), \tag{4.5}$$

where w_i is the weight for each estimator $q(S_i)$ and is constrained by $\sum_{i=1}^{k} w_i = 1$. Note that we may allocate no budget to a sampler. In this case we remove its sample from ψ.

The variance of the combined estimator is computed as [1]:

$$\mathsf{var}\Big(q(\psi)\Big) = \sum_{i=1}^{k} w_i^2 \cdot \mathsf{var}(q(S_i)) = \sum_{i=1}^{k} w_i^2 \cdot \frac{\mathsf{var}(\mathcal{D}_i)}{n_i} \tag{4.6}$$

Recall that our goal is to minimize the estimation error (confidence interval) using the combined estimator. Since the confidence interval can be computed from variance (see Section 4.2.1), our goal is equivalent to minimize the variance of the combined estimator.

We call this problem the sampler combination problem, as formalized in Problem 3.

Problem 3 (Sampler Combination). *Given a set of k samplers, a query q, and a total budget n, the goal of the sampler combination problem is to decide the sample size n_i for each sampler such that the combined estimator has the minimized variance:*

$$\begin{aligned} \underset{n_1,\ldots,n_k,w_1,\ldots,w_k}{\arg\min} \quad & \sum_{i=1}^{k} w_i^2 \cdot \frac{\mathit{var}(\mathcal{D}_i)}{n_i} \\ \textit{subject to} \quad & \sum_{i=1}^{k} n_i = n \\ & n_i \geq 0, \quad \textit{for all } i \in [1, k] \\ & \sum_{i=1}^{k} w_i = 1 \end{aligned} \tag{4.7}$$

4.3 Sampler Combination Framework

In this section, we discuss how to solve the sampler combination problem and present the sampler combination framework.

4.3.1 Optimal Weight Allocation

The sampler combination problem consists of two sub-problems:

1. Budget Allocation. How should we decide the sample size n_i for each sampler?
2. Weight Allocation. How should we decide the weight w_i for each estimator?

[1]Note that each sampler draws sample independently.

Let us first consider the second problem by assuming $n_1, n_2, \cdots, n_k$ have been decided. The problem can be formalized as follows:

$$\begin{aligned} \underset{w_1, w_2, \ldots, w_k}{\arg\min} \quad & \sum_{i=1}^{k} w_i^2 \frac{\mathsf{var}(\mathcal{D}_i)}{n_i} \\ \text{subject to} \quad & \sum_{i=1}^{k} w_i = 1, \end{aligned}$$

where $n_1, n_2, \cdots, n_k$ are constant values.

By applying Lagrange's method of multipliers[53], we can get the optimal weight allocation: $w_i = \frac{\frac{n_i}{\mathsf{var}(\mathcal{D}_i)}}{\sum_{j=1}^{k} \frac{n_j}{\mathsf{var}(\mathcal{D}_j)}}$. By incorporating the optimal weight into Equation (4.7), the sampler combination problem is reduced as follows:

$$\begin{aligned} \underset{n_1, \ldots, n_k}{\arg\min} \quad & \frac{1}{\sum_{i=1}^{k} \frac{n_i}{\mathsf{var}(\mathcal{D}_i)}} \\ \text{subject to} \quad & \sum_{i=1}^{k} n_i = n \\ & n_i \geq 0, \quad \text{for all } i \in [1, k] \end{aligned} \tag{4.8}$$

Thus, the key to solve the sampler combination problem is how to solve the budget allocation problem.

4.3.2 Exploration and Exploitation Trade-off

We first propose the optimal solution to the budget allocation problem and prove its optimality. However, please note that this solution cannot be achieved in practice, hence we call it *ideal* solution. We then explain the reason and find an interesting trade-off between the exploration and exploitation to develop a practical solution.

Ideal Solution. Given a query, different samplers produce estimators with different qualities. Intuitively, allocating more budget to a 'good' sampler will lead to a more accurate combined estimator. This intuition inspires us to consider an extreme case which allocates all the budget to the 'best' sampler.

Specifically, we use $\mathsf{var}(\mathcal{D}_i)$ (see Definition 5) to measure how 'good' a sampler is. The smaller $\mathsf{var}(\mathcal{D}_i)$ is, the better the sampler is. Let $\mathbb{S}_{i^*}$ denote the best sampler, i.e., $i^* = \arg\min_{i \in [1,k]} \mathsf{var}(\mathcal{D}_i)$.

The ideal solution allocates all the budget to $\mathbb{S}_{i^*}$ and allocates zero budget to $\mathbb{S}_i$ (for $i \neq i^*$). That is, for each $i \in [1, n]$, we have:

$$n_i = \begin{cases} n, & \text{if } i = i^* \\ 0, & \text{otherwise} \end{cases}$$

Optimality. We then prove that the ideal solution is the optimal solution of the budget allocation problem (formalized in Equation (4.8)), as shown in Lemma 7.

Lemma 7. *Given a set of k samplers, a query q, and a total budget n, the optimal budget allocation is to allocate all the budget n to the best sampler $\mathbb{S}_{i^*}$.*

Proof. All the proofs in this work can be found in Appendix B. □

Exploration vs. Exploitation. Although we get the optimal solution of the budget allocation problem, this solution is impractical. The reason is that it requires the knowledge of the best sampler in advance, while computing each sampler quality $\mathsf{var}(D_i)$ requires scanning the full data and against the purpose the sampling. To solve this problem, we develop a framework to approach the ideal solution, and it can allocate most of the budget to the best sampler without knowing which sampler is the best. More specifically, we use a sample to estimate $\mathsf{var}(D_i)$ and define *empirical best sampler* as the sampler with the highest estimated sampler quality. In this way, the sample budget can be allocated with two different purposes: i) *Exploration:* it is allocated to estimate each sampler quality in order to find the best sampler; ii) *Exploitation:* it is allocated to the *empirical* best sampler in order to enhance the combined estimator.

There is an interesting trade-off between exploration and exploitation. When allocating more budget to explore sampler quality, it is more likely to find the true best sampler. However, there will be less budget left for the empirical best sampler to enhance the combined estimator. On the other hand, when allocating less budget to explore sampler quality, it is more likely to estimate sampler quality incorrectly and regard a bad sampler as the best sampler. As a result, most of the budget could be allocated to this bad sampler.

4.3.3 Model as Multi-Armed Bandit

The well-known Multi-Armed Bandit (MAB) problem also faces the exploration and exploitation trade-off [156]. One natural question is that can we borrow some ideas from MAB to solve our problem? In this subsection, we first introduce some background about MAB and then present how to model our problem as MAB. Finally, we justify why this modeling makes sense.

Background. MAB is a classical reinforcement learning problem that studies the exploration and exploitation trade-off. Consider a slot machine with k arms. A player needs to decide which arm to pull at each round. If arm i is pulled, it will return a random reward, usually in $[0, 1]$, from an unknown reward distribution $\mathcal{D}_i$ specific to arm i. The goal of the player is to maximize the total reward, or minimize the regret (will be defined later), in n rounds.

Let $\mu_i = \mathbb{E}[\mathcal{D}_i]$ and $\mu^* = max_{i \in 1 \dots k}\{\mu_i\}$. Clearly, the optimal strategy is to always pull the arm with the highest expected reward. However, since we do not know the best arm in advance, there exists a 'regret' of the actual pulling strategy compared to the optimal strategy. The regret over n rounds, denoted by R_n, is defined as the difference of the total rewards achieved by the optimal pulling strategy and by the actual pulling strategy, i.e.,

$$R_n = \mu^* \cdot n - \sum_{t=1}^{n} \mu_{I_t}, \tag{4.9}$$

where I_t is the chosen arm in round t. The expected regret is $\mathbb{E}[R_n] = \mu^* \cdot n - \sum_{t=1}^{n} \mathbb{E}[\mu_{I_t}]$. We denote $\Delta_i = \mu^* - \mu_i$ by the reward gap between an arm i and the best arm, and rewrite $\mathbb{E}[R_n]$ in terms of Δ_i:

$$\mathbb{E}[R_n] = \sum_{i=1}^{k} \Big(\Delta_i \cdot \mathbb{E}[T_i(n)] \Big), \tag{4.10}$$

where $T_i(n)$ is the number of times that an arm i is pulled over n rounds, and clearly, they satisfy $\sum_{i=1}^{k} T_i(n) = n$.

The goal is to find a pulling strategy to minimize $\mathbb{E}[R_n]$. For a given arm i, if it is an optimal arm, i.e., $\mu^* = \mu_i$, then we have $\Delta_i = 0$, thus no matter how many times the optimal arm is pulled, there is no impact on $\mathbb{E}[R_n]$. Thus, only the number of times that each sub-optimal arm is pulled will affect $\mathbb{E}(R_n)$. There are several pulling strategies proposed in the MAB literature [156]. They provide good theoretical guarantees. When using these strategies, the number of times that each sub-optimal arm is pulled can be upper-bounded by $O(\ln n)$.

Modeling. We next discuss how to model the sampler combination problem as MAB. Each sampler can be regarded as an arm in MAB. At each round, we need to pick up one sampler and draw a small sample from it. This can be thought of as picking up an arm and pulling it to get a random reward.

The *key difference* between the two problems is the definition of the regret (i.e., the objective function). For MAB, as shown in Equation (4.9), the regret is a linear combination of the observed random reward of each round. For our problem, the regret measures the difference of the estimator variances between an allocation strategy and the optimal strategy (i.e., allocating all the budget to the best sampler), which is defined as

$$R_n = \frac{1}{\sum_{i=1}^{k} \frac{n_i}{\mathsf{var}(\mathcal{D}_i)}} - \frac{\mathsf{var}(\mathcal{D}_{i^*})}{n}, \tag{4.11}$$

where i^* is the index of the best sampler, n_i is the size allocated to sampler $\mathbb{S}_i$, and n is the total sample size.

Justification. Although our problem has a different objective function than the MAB problem (Equation (4.9) vs. Equation (4.11)), the two objective functions share two common properties, which make it possible to apply MAB-based approaches to solve our problem. First, their optimal solutions are the same, i.e., allocating all the budget to the best sampler (or arm), which lead to zero regret.

Property 1. *The optimal solution to the sampler combination (or MAB) problem is to allocate all the sampling (pulling) budget to the best sampler (or arm).*

In reality, however, the optimal solution of sampler combination problem cannot be achieved since the best sampler is unknown. This is similar to MAB setting where the arm quality is unknown and needs to be estimated from each pull. It inspires us to solve the problem in an iterative framework similar to MAB: explore the sampler quality and exploit budgets to the empirically best sampler.

We also interestingly observe that for both objective functions, the optimal action is independent of what actions have been taken in the previous iterations. That is, it is always optimal to choose the best sampler (or arm) at every individual iteration, regardless of which samplers were chosen in the previous iterations.

Property 2. *At each iteration, the optimal action for the sampler combination (or MAB) problem is to choose the best sampler (or arm), which is independent of historical actions.*

This property implies that MAB and SamComb follows the same principal to take action in each iteration: choose the best one in each iteration. Therefore, if we apply an MAB-like strategy, the action it takes may also lead us to its guarantee: most budgets are allocated to the best sampler, which can help achieve our goal.

4.3.4 Framework

Since our problem and MAB shares the same goal, i.e., choose the best sampler (arm) as many times as possible. Thus, it makes sense to model our problem as MAB. To this end, we propose SamComb, an MAB-based sampler combination framework.

We first introduce the existing MAB framework. Initially, the framework pulls each arm once, to get an initial estimation of the arm quality. After that, it decides which arm to pull in an iterative scheme. The fundamental challenge is how to balance the exploration and exploitation trade-off. Well-known strategies, such as ϵ-greedy and UCB [49], are proposed to handle the trade-off systematically, and guarantee that the best arm is pulled as many times as possible.

The framework of SamComb is shown in Figure 4.2. It is an iterative framework based on MAB, and consists of three phases:

1. *Initialization Phase:* In the initialization phase, SamComb draws an initial batch of tuples from each sampler, to get an initial estimation of sampler quality.

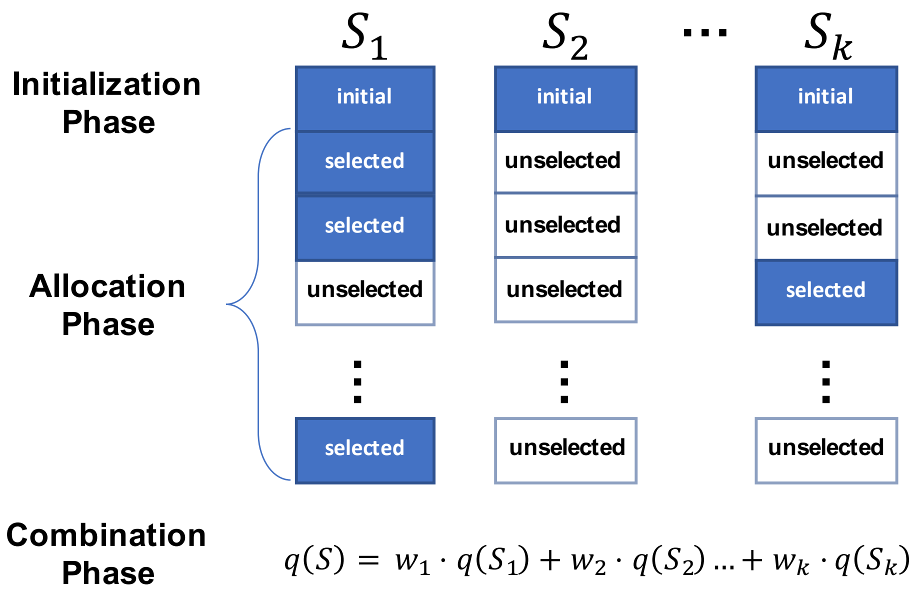


Figure 4.2: The SamComb Framework

2. *Allocation Phase:* In the allocation phase, at each iteration, SamComb decides which sampler to select in order to achieve the best trade-off between exploration and exploitation and then draws a small batch of tuples from it. We extend the ϵ-greedy and UCB strategies from MAB and prove their theoretical guarantees in Section 4.4. The allocation phase stops once the budget is exhausted.

3. *Combination Phase:* After the allocation phase, suppose each sampler is allocated a sample of n_i tuples such that $\sum_{i=1}^{k} n_i = n$. In the combination phase, SamComb computes an estimator $q(S_i)$ from each sampler and combines them (by computing a weighted average) into the final estimator. We propose two weight allocation approaches in Section 4.5.

4.4 Allocation Phase

We first present the ϵ_t-greedy strategy in Section 4.4.1 and then the Lower Confidence Bound (LCB) strategy in Section 4.4.2. We prove that both strategies can guarantee that the allocated sample size to any sub-optimal sampler is at most $O(\ln n)$, where n is the total sampling budget.

4.4.1 ϵ_t-greedy

ϵ_t-greedy [49] is a simple but powerful approach in MAB. It controls the exploration and exploitation trade-off through a parameter ϵ_t: at each iteration t, it pulls a random arm with a probability of ϵ_t (exploration), and pulls the empirical best arm with a probability of $1 - \epsilon_t$ (exploitation).

A simple approach is to set ϵ_t to a fixed value, i.e., keeping the same trade-off at every iteration. However, it contradicts the intuition that we should explore more in early iterations to look for the best arm and then exploit more in later iterations once the empirical best arm is more likely to be the true best arm.

A more sophisticated approach is proposed to address this issue [49]. It progressively decreases ϵ_t as iteration t increases. I.e., $\epsilon_t = \min\{1, \frac{ck}{d^2 t}\}$, where t refers to the t-th iteration, k is the number of arms, and c and d are user-specified parameters. It is proved that when $c > 5$, and d is upper-bounded by the reward gap between the best arm and the second best arm (i.e., $0 < d \leq \min_{i:\mu_i<\mu^*}\{\Delta_i\}$, where Δ_i is the reward gap between the best arm and arm i), ϵ_t-greedy achieves a logarithmic regret [49]. It is much better than the strategy of using a fixed ϵ.

We extend the ϵ_t-greedy strategy to solve our problem. Algorithm 1 shows the pseudo-code. The main challenges are i) how to identify the empirical best sampler, and ii) how to prove the theoretical guarantee of our ϵ_t-greedy strategy.

Empirical Best Sampler. We start with the first challenge. At each iteration, we need to identify the empirical best sampler, which requires estimating the sampler quality. In MAB, the arm quality is estimated by averaging the random reward for each pull of the arm. However, directly applying this approach does not work. This is because MAB only pulls the arm once in each iteration to get an estimation (i.e., the random reward), but the sampler quality is a variance and cannot be estimated using a single tuple. To get multiple tuples, one may consider reusing tuples from previous iterations. Unfortunately, this approach loses the independence of the estimations between different iterations, which makes Hoeffding's inequality fail and lose the guarantee provided by MAB. To solve the above issue, we propose a batch-based approach: for each iteration, we pull a batch of tuples to estimate the sampler quality, then we average the estimations of different iterations to get the final estimation. In this way, we can estimate the sampler quality in each iteration, and also preserve the independence between iterations.

Let ΔS denote a batch of tuples randomly drawn from sampler $\mathbb{S}$. The estimated sampler quality using ΔS is computed as [2]:

$$\mathbb{S}.\mathsf{batchEst} = \frac{1}{|\Delta S| - 1} \cdot \sum_{i=1}^{|\Delta S|} (\frac{y_i}{p_i} - q(\Delta S))^2 \tag{4.12}$$

Note that ΔS could be empty if the current budget is not allocated to the sampler. In this case, $\mathbb{S}.\mathsf{batchEst}$ is set to 0.

[2]See Theorem 4.2.3 in [154]. Note that it needs to be multiplied by n to get our estimator.

Algorithm 1: ϵ_t-greedy strategy

```
Input  : A set of samplers $\Psi = \{\mathbb{S}_1, \mathbb{S}_2, \cdots, \mathbb{S}_k\}$, budget $n$, batch size $b$, parameters $c$ and $d$
Output: A set of samples $\psi = \{S_1, S_2, \cdots, S_k\}$
1  # Initialization Phase
2  for each sampler $\mathbb{S}$ in $\Psi$ do
3      $\Delta S$: Draw a batch of $b$ tuples from $\mathbb{S}$ ;
4      $\mathbb{S}.\mathsf{batchEst} = \frac{1}{|\Delta S|-1} \cdot \sum_{i=1}^{|\Delta S|}(\frac{y_i}{p_i} - q(\Delta S))^2$ ;
5      $\mathbb{S}.\mathsf{batchEstSum} = \mathbb{S}.\mathsf{batchEst}$ ;
6      $\mathbb{S}.\mathsf{batchNum} = 1$;
7      $\mathbb{S}.\mathsf{sample} = \Delta S$
8  end
9  # Allocation Phase
10 for $t = 1$ to $n/b$ do
11     $\epsilon_t = \mathsf{min}\{1, \frac{ck}{d^2 t}\}$ ;
12     if $rand() > \epsilon_t$ then
13         $\mathbb{S}^* = \arg\min_{\mathbb{S}\in\Psi} \frac{\mathbb{S}.\mathsf{batchEstSum}}{\mathbb{S}.\mathsf{batchNum}}$ ;
14     end
15     else
16         $\mathbb{S}^*$ = a random sampler from $\Psi$;
17     end
18     $\Delta S$: Draw a batch of $b$ tuples from $\mathbb{S}^*$ ;
19     $\mathbb{S}^*.\mathsf{batchEst} = \frac{1}{|\Delta S|-1} \cdot \sum_{i=1}^{|\Delta S|}(\frac{y_i}{p_i} - q(\Delta S))^2$ ;
20     $\mathbb{S}^*.\mathsf{batchEstSum} += \mathbb{S}^*.\mathsf{batchEst}$ ;
21     $\mathbb{S}^*.\mathsf{batchNum} += 1$;
22     $\mathbb{S}^*.\mathsf{sample} += \Delta S$ ;
23 end
24 $\psi = \{\mathbb{S}.\mathsf{sample} \mid$ for each $\mathbb{S} \in \Psi\}$ ;
25 return $\psi$
```

After getting an estimation of the sampler quality from each batch, we then average them to get the final estimation. The sampler with the minimal average value is identified as the empirical best sampler at the t-th iteration, i.e.,

$$\mathbb{S}^* = \arg\min_{\mathbb{S}\in\Psi} \frac{\sum_{i=0}^{t} \mathbb{S}.\mathsf{batchEst}_i}{\mathbb{S}.\mathsf{batchNum}},$$

where batchNum is the total number of batches that have been allocated to this sampler after t iterations.

Theoretical Guarantee. We theoretically analyze how well our ϵ_t-greedy strategy works compared to the optimal allocation strategy, which allocates all the budget to the best sampler. In MAB, ϵ_t-greedy is proved to have a logarithmic regret bound, and a sub-optimal arm is pulled at most $\mathcal{O}(\ln n)$, given a total pulling times of n. We find that a similar theoretical guarantee also holds in our problem. Similar to MAB, the theoretical guarantee requires the estimation from each batch bounded. We use u_i to denote the bound, which is computed as $u_i = \max_{j \in 1 \dots N}\{(\frac{y_j}{p_{ij}})^2\} - \min_{j \in 1 \dots N}\{(\frac{y_j}{p_{ij}})^2\}$. I.e.,

$$0 \leq \mathbb{S}.\mathsf{batchEst} \leq u_i$$

In Lemma 8, we prove that the budget allocated to a sub-optimal sampler is bounded by $\mathcal{O}(\ln n)$, given a total budget n, when $c > 5$ and $0 < d \leq \min_{i \neq i^*} \frac{\Delta_i}{u_i}$, where Δ_i is the quality gap between the best sampler and sampler i.

Lemma 8. *Given a total budget n, if ϵ_t-greedy is running with parameters $c > 5$ and $0 < d \leq \min_{i \neq i^*} \frac{\Delta_i}{u_i}$, then the budget allocated to any sub-optimal sampler is at most $\mathcal{O}(\ln n)$.*

4.4.2 LCB

We propose the Lower Confidence Bound (LCB) strategy in this section. It is inspired by the Upper Confidence Bound (UCB) [49] strategy in MAB.

UCB does not only estimate arm quality but also its confidence bound, where the confidence bound represents the uncertainty of the estimation. The key idea of UCB is to be optimistic about the uncertainty of the estimation. More specifically, it pulls the arm with the highest upper confidence bound of the estimation of arm quality, i.e., the arm with the highest quality in the optimistic case. For the selected arm, there exists two cases: 1) if it is the best arm, then this is exactly what we want; 2) if it is not the best sampler, then pulling it will increase our confidence on its sampler-quality estimation (i.e., decreasing the size of the confidence bound). As a result, it is less likely to be selected in future. Hence, UCB decreases the probability of pulling a sub-optimal arm, and exploits more and more on the best arm as iteration increases. It has been proved that UCB achieves a logarithmic bound on regret [49].

We extend UCB to our problem. Since the smaller the $\mathsf{var}(\mathcal{D})$, the better the sampler quality, we propose a strategy named Lower Confidence Bound (LCB). Similar to the reason described in Section 4.4.1, we cannot directly apply UCB and we need to preserve the independence between estimations of each iteration, hence we also adopt a batch-based approach rather than drawing a single tuple in each iteration. The pseudo code is shown in Algorithm 2. LCB shares a similar idea with UCB: at each iteration, it estimates the sampler quality and computes the confidence bound of the estimation. Then, it allocates the budget to the sampler with the lowest lower confidence bound, i.e., the sampler with the highest quality in the optimistic case. LCB faces two new challenges: 1) how to compute the lower confidence bound of the estimation of sampler quality? 2) can we derive a similar theoretical guarantee like Lemma 8 for LCB?

Lower Confidence Bound. We start with the first challenge. Essentially, the confidence interval measures how far a random variable is from its expectation, which could be computed from concentration inequalities. For UCB, it regards each random reward as a random

Algorithm 2: LCB strategy

```
Input  : A set of samplers $\Psi = \{\mathbb{S}_1, \mathbb{S}_2, \cdots, \mathbb{S}_k\}$, budget $n$, batch size $b$, bound $u_i$
Output: A set of samples $\psi = \{S_1, S_2, \cdots, S_k\}$
1  # Initialization Phase is the same as Algorithm 1
2  # Allocation Phase
3  for $t = 1$ to $n/b$ do
4      $\mathbb{S}^* = \arg\min_{\mathbb{S}_i} \frac{\mathbb{S}_i.\mathsf{batchEstSum}}{\mathbb{S}_i.\mathsf{batchNum}} - u_i\sqrt{\frac{\ln t}{\mathbb{S}_i.\mathsf{batchNum}}}$
5      $\Delta S$: Draw a batch of $b$ tuples from $\mathbb{S}^*$ ;
6      $\mathbb{S}^*.\mathsf{batchEst} = \frac{1}{|\Delta S|-1} \cdot \sum_{i=1}^{|\Delta S|}(\frac{y_i}{p_i} - q(\Delta S))^2$ ;
7      $\mathbb{S}^*.\mathsf{batchEstSum} += \mathbb{S}^*.\mathsf{batchEst}$ ;
8      $\mathbb{S}^*.\mathsf{batchNum}$ += 1;
9      $\mathbb{S}^*.\mathsf{sample} += \Delta S$ ;
10 end
11 $\psi = \{\mathbb{S}.\mathsf{sample} \mid$ for each $\mathbb{S} \in \Psi\}$ ;
12 return $\psi$
```

variable and applies Hoeffding inequality, as stated in Lemma 9, to compute the confidence interval.

Lemma 9 (Hoeffding Inequality). *Let $X_1, \dots, X_n$ be independent random variables bounded by the interval $[a_i, b_i]$: $a_i \le X_i \le b_i$. Let $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$, then Hoeffding Bound states that:*

$$Pr(|\overline{X} - E[\overline{X}]| \ge \lambda) \le 2exp(-\frac{2n^2\lambda^2}{\sum_{i=1}^{n}(b_i - a_i)^2}) \tag{4.13}$$

In Lemma 9, $(\overline{X} - \lambda, \overline{X} + \lambda)$ represents the confidence interval, and $2exp(-\frac{2n^2\lambda^2}{\sum_{i=1}^{n}(b_i-a_i)^2})$ refers to the confidence level. To extend UCB to LCB, we also adapt the Hoeffding inequality. Note that Hoeffding inequality requires the random variables to be bounded and independent from each other. Hence, we estimate the sampler quality for each batch, as described in Section 4.4.1, and regard that estimation (i.e., $\mathbb{S}.\mathsf{batchEst}$) as a random variable. Each random variable is bounded by the interval of $[0, u_i]$. We then use the average of all random variables as the final estimation of the sampler quality, and denote it by $\hat{\mathsf{var}}(\mathcal{D})_t$, i.e.,

$$\hat{\mathsf{var}}(\mathcal{D})_t = \frac{\sum_{i=1}^{t} \mathbb{S}.\mathsf{batchEst}_i}{\mathbb{S}.\mathsf{batchNum}}$$

Next we compute the confidence interval of $\hat{\mathsf{var}}(\mathcal{D})_t$ by applying Hoeffding inequality. The variable number n in Hoeffding inequality (Equation (4.13)) is our batch number, i.e., $\mathbb{S}.\mathsf{batchNum}$. We use σ_t to represent the confidence level at iteration t, and replace λ_t with σ_t in Equation (4.13). Then the confidence interval is $\lambda_t = u_i\sqrt{\frac{\ln\frac{2}{\sigma_t}}{2\mathbb{S}.\mathsf{batchNum}_t}}$. I.e.[3],

$$Pr\left(|\hat{\mathsf{var}}(\mathcal{D})_t - \mathsf{var}(\mathcal{D})| \ge u_i\sqrt{\frac{\ln\frac{2}{\sigma_t}}{2\mathbb{S}.\mathsf{batchNum}_t}}\right) \le \sigma_t \tag{4.14}$$

[3]Note that $\hat{\mathsf{var}}(\mathcal{D})_t$ is an unbiased estimator and we have $E[\hat{\mathsf{var}}(\mathcal{D})_t] = \mathsf{var}(\mathcal{D})$.

```
Algorithm 3: AdaptiveLCB strategy
  Input  : Sampler set $\Psi = \{\mathbb{S}_1, \mathbb{S}_2, \cdots, \mathbb{S}_k\}$, budget $n$, batch size $b$
  Output: A set of samples $\psi = \{S_1, S_2, \cdots, S_k\}$
1 Insert "$u_i = \mathbb{S}_i.\mathsf{batchEst}$" between Line 1 and 2 in Algorithm 2
2 Insert "$u_{i^*} = max(u_{i^*}, \mathbb{S}^*.\mathsf{batchEst})$" between Line 9 and Line 10 in Algorithm 2
```

We set σ_t to $\frac{2}{t^2}$ to make $\sum \sigma_t$ converged. Hence we compute the lower confidence bound of each sampler $\mathbb{S}$ at iteration t as follows:

$$\mathsf{lcb}_t = \hat{\mathsf{var}}(\mathcal{D})_t - u_i \sqrt{\frac{\ln t}{\mathbb{S}.\mathsf{batchNum}_t}} \tag{4.15}$$

It satisfies:

$$Pr\left(\mathsf{lcb}_t \geq \mathsf{var}(\mathcal{D})\right) \leq \sigma_t \tag{4.16}$$

That is, with high probability, the lower confidence bound is no larger than the real sampler quality.

Theoretical Guarantee. We discuss how to address the second challenge, i.e., what theoretical guarantee can we get for LCB? In MAB problem, UCB strategy is proved to pull a sub-optimal sampler at most $O(\ln n)$ times, and achieve a logarithmic regret. We find that a similar bound also holds for LCB. In Lemma 10, we show that the budget allocated to a sub-optimal sampler is bounded by $O(\ln n)$.

Lemma 10. *Given a total budget n, if applying the LCB strategy, the budget allocated to any sub-optimal sampler is at most $\mathcal{O}(\ln n)$.*

AdaptiveLCB. In LCB, we need u_i, which is the bound of the sampler-quality estimation from each batch, such that the Hoeffding inequality can be applied and the theoretical analysis holds. However, u_i could be very large. When the budget is limited, a large u_i may make LCB spend too much budget on exploration, thus decreasing the overall performance.

To solve this issue, we propose a variation of LCB named AdaptiveLCB. Instead of using a large u_i that can bound all possible estimations from each batch (both historical and future estimation), we set u_i adaptively such that it can bound all the historical estimations. More specifically, we record the current maximal value of the sampler-quality estimation in each iteration, and set it to u_i. Algorithm 3 shows the pseudo code of AdaptiveLCB. We only need to add two lines of code into Algorithm 2.

Example 6 illustrates how AdaptiveLCB works.

Example 6. *Suppose we have two samplers $\mathbb{S}_1$ and $\mathbb{S}_2$, and the batch size is 100. We first draw a batch of 100 tuples from each sampler, and use it to estimate the sampler quality. Suppose the estimated quality for $\mathbb{S}_1$ and $\mathbb{S}_2$ are 10000 and 20000, respectively. Since u_i is the adaptive bound of the sampler-quality estimation, u_1 and u_2 are initialized as 10000 and 20000, respectively.*

In the first iteration, the lower confidence bound for $\mathbb{S}_1$ *and* $\mathbb{S}_2$ *are* $10000 - 10000\sqrt{\frac{\ln 1}{1}} = 10000$ *and* $20000 - 20000\sqrt{\frac{\ln 1}{1}} = 20000$*, respectively (see Equation* (4.15)*). Hence,* $\mathbb{S}_1$ *is the current empirical best sampler. We draw a batch of 100 tuples from* $\mathbb{S}_1$ *and use it to compute a sampler-quality estimation. Suppose it is 50000. Then the average sampler-quality estimation of* $\mathbb{S}_1$ *is updated as* $\frac{10000+50000}{2} = 30000$ *and* u_1 *is updated as* $\max\{u_1, 50000\} = 50000$*.*

In the second iteration, the lower confidence bound for $\mathbb{S}_1$ *and* $\mathbb{S}_2$ *are* $30000 - 50000\sqrt{\frac{\ln 2}{2}} = 565$ *and* $20000 - 20000\sqrt{\frac{\ln 2}{1}} = 3349$*, respectively.* $\mathbb{S}_1$ *is still the empirical best sampler, thus we allocate a batch to* $\mathbb{S}_1$*.*

Repeat the iterative process until the budget is exhausted.

4.5 Combination Phase

Once the budget is exhausted, the allocation phase is finished. Now SamComb enters the combination phase. Let $\psi = \{S_1, S_2, \cdots, S_k\}$ denote the sample set derived from our allocation strategy (ϵ-greedy or LCB). The goal of the combination phase is to assign a weight to each sampler such that the variance, $\mathsf{var}(q(\psi))$, of the combined estimator is minimized.

Section 4.3.1 presents the optimal solution to this weight allocation problem. However, the optimal weight requires knowing $\mathsf{var}(D_i)$ (for each $i \in [1, k]$), which is *not* available in reality. One straightforward solution is to estimate $\mathsf{var}(\mathcal{D}_i)$ using the allocated sample and then apply this optimal weight allocation. We call such approach *pseudo-optimal allocation*, where the weight is computed as follows:

$$w_i = \frac{n_i / \hat{\mathsf{var}}(\mathcal{D}_i)}{\sum_{j=1}^{k} n_j / \hat{\mathsf{var}}(\mathcal{D}_j)}, \tag{4.17}$$

where $\hat{\mathsf{var}}(\mathcal{D}_i)$ is an estimation of $\mathsf{var}(\mathcal{D}_i)$ based on S_i.

Let ψ_* denote the sample set derived from the optimal budget allocation strategy, i.e., allocating all the budget to the best sampler. Let $\mathsf{var}(q(\psi_*))$ denote the corresponding variance. We use the following formula to measure the gap of SamComb to the optimal strategy:

$$gap(q(\psi)) = \frac{\mathsf{var}(q(\psi)) - \mathsf{var}(q(\psi_*))}{\mathsf{var}(q(\psi_*))} \tag{4.18}$$

Lemma 11 proves that SamComb is asymptotically optimal under the pseudo-optimal allocation, when $\hat{\mathsf{var}}(\mathcal{D}_i) = \mathsf{var}(\mathcal{D}_i)$ for each $i \in [1, k]$.

Lemma 11. *Under the assumption that* $\hat{\mathsf{var}}(\mathcal{D}_i) = \mathsf{var}(\mathcal{D}_i)$ *for each* $i \in [1, k]$*, our framework SamComb, which uses* ϵ*-greedy or LCB for budget allocation and uses the pseudo-optimal allocation strategy for weight allocation, is asymptotically optimal.*

We next explore an alternative weight allocation strategy to relax the assumption in Lemma 11. Obviously, a good sampler should receive a higher weight than a bad one. Thus, the key challenge is how to find an alternative way to assess sampler quality. We observe that after the budget allocation phase , the better the sampler, the larger sample size it tends to receive, since this is what our budget allocation strategy tries to optimize for. Thus, the received sample size is an indirect way to assess sampler quality. Based on this idea, we propose *proportional-to-size allocation*,

$$w_i = \frac{n_i}{\sum_{i=1}^{k} n_i}, \tag{4.19}$$

where n_i is the sample size received by the sampler i after the budget allocation phase.

Lemma 12 proves that SamComb is asymptotically optimal under the proportional-to-size allocation.

Lemma 12. *Our framework SamComb, which uses ϵ-greedy or LCB for budget allocation and uses the proportional-to-size allocation strategy for weight allocation, is asymptotically optimal.*

We experimentally compare the two weight allocation strategies and find that they have similar performance. Since the proportional-to-size allocation provides a nice theoretical guarantee, SamComb uses it by default.

4.6 Extensions

In this section, we discuss how to extend our framework to support other aggregate functions, group-by queries, and data updates.

Other Aggregate Functions. Previously we mainly discussed `SUM`-like query. Actually SamComb can be extended to support more aggregate functions, such as `MIN`, `MAX` and `PERCENTILE` query.

We first introduce how to estimate the answer using a single sampler. Suppose a tuple is sampled with a probability of 0.1, then it approximately represents 10 such tuples in the population. In this way, we can "reconstructed" the population and issue the original query over the "reconstructed" population to get an estimation. This idea is similar to the plug-in approach in [122]. Note that the "reconstructed" usually happened virtually. For many aggregation functions we do not need to actually rebuild the population. Take q-percentile (e.g., $q = 0.9$ represents the 90-th percentile) query as an example. As each tuple t in the sample represents $1/p_t$ tuples in the "reconstructed" population, where p_t is drawn probability of tuple t, we denote its weight as $1/p_t$. Then we sort the sampled tuples by their values, and accumulating the weights. The q-percentile query can be estimated as the largest value when the accumulated weights is no larger than q multiply by the total weights.

Now we discuss the case of multiple samplers. To be supported by SamComb, the aggregate function needs to specify: 1) how to allocate the budget in each iteration and 2) how to combine the results from multiple samplers.

For the first question, since most aggregate functions can not compute the confidence bound easily or efficiently, we adopt the ϵ-greedy strategy by default. To compare samplers' qualities, by default we compare the selectivity in their sample. One may also use different approaches for different functions. E.g., for `MAX` query, we can simply compare the max value in the sample (exclude the tuples which do not satisfy the predicate), since the sampler with a larger max value definitely makes a better estimation.

For the second question, since linear combination (by w_i) does not work for all the aggregate functions, a similar idea of reconstructing the population can be applied to the general case. For example, to process a median query, we can reconstruct multiple populations from different samplers, and then merge them and choose the median value of the merged reconstructed population.

Group-by Queries. To process group-by query, we will maintain the related information for each group and apply previously discussed approaches to estimate each group. Then, the left question is how to select the sampler in each iteration. To answer this question, we consider minimizing the max error among groups. For SUM-like queries, we use the estimator variance to measure the group error and select the sampler that has the highest quality for the group with the max error. For other aggregate queries (e.g., percentile), we use # of tuples to measure the group error. Let g be the group with the minimal tuples, we then select the sampler that are more likely to contain tuples in g.

Data Update. In this work we focused on insert-only update, like many existing works [131]. When data is updated, SamComb will maintain the pre-computed samples. Let D_Θ be the stale data, S_Θ be the stale sample created by a sampler $\mathcal{S}$, and D_Δ be inserted data, respectively. Now, the goal is to maintain S_Θ such that it is equivalent to a sample with the same size drawn by $\mathcal{S}$ from $D_\Theta \cup D_\Delta$. To make the maintenance efficient, the high level idea is to replace some tuples in S_Θ with a sample drawn from D_Δ. I.e., the new sample consists of two parts: the kept tuples in S_Θ, denoted as S'_Θ; and the drawn sample from D_Δ, denoted as S_Δ. For example, a uniform sampler draws each tuple with probability $p_i = \frac{1}{N}$. In sample S_Θ, each tuple is drawn with probability $\frac{1}{N_\Theta}$. After data is updated, each tuple should be drawn with probability $\frac{1}{N_\Theta + N_\Delta}$. Hence, we keep each tuple in S_Θ with probability $\frac{N_\Theta}{N_\Theta + N_\Delta}$ to get S'_Θ, and draw each tuple in D_Δ with probability $\frac{1}{N_\Theta + N_\Delta}$ to get S_Δ. Finally, the updated sample is $S'_\Theta \cup S_\Delta$.

4.7 Experiment

We evaluate SamComb using both synthetic and real datasets. The experiments aim to answer the following questions:

- What should be the best setting for SamComb?
- Is it necessary to combine multiple samplers?
- Is bandit-based better than heuristic-based?
- How does SamComb perform in various settings?

4.7.1 Experimental Setup

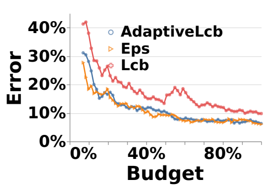
Datasets. 1) *TPCS* is a synthetic dataset generated from a variation of the TPC-H benchmark [67]. We set parameters skewness $z = 2$ and scale $s = 1$, and focused on the lineitem table, which contains 6 million rows and 16 columns. We also generated a larger dataset using scale $s = 10$ to test the end-to-end performance. 2) Loan [2] is a real-world peer-to-peer loan dataset. It concatenated historical loans from Prosper and Lending Club from 2013 to 2018. The dataset contains 3 million rows and 18 columns with a wide variety of data distributions.

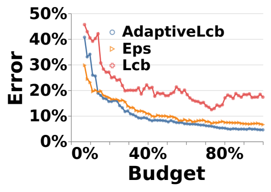
Samplers. We created a uniform sampler, a stratified sampler on `l_returnflag`, and a measure-biased sampler on `l_extendedprice` for *TPCS* dataset, and created a uniform sampler, a stratified sampler on `grade`, and a measure-biased sampler on `principal_balance` for *Loan* dataset. Note that a few tuples may have a very large value and thus be selected too many times by the measure-biased sampler, hence we will binnizate the measure column before computing the sampling probability.

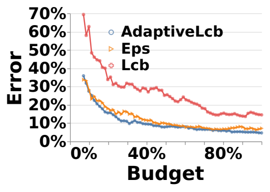
Queries. The population parameters that we aim to estimate are in the form of `SELECT SUM(A) FROM table WHERE Condition(B1) and Condition(B2), ..., Condition(Bk)`. The query selectivity is between 0.1% and 1% of the population. For each query, the aggregation column `A` and each condition column `Bi` are randomly selected, and the number of condition columns is a random number between 1 and 5. `Condition(Bi)` is in the form of `x` $\leq$ `Bi` $\leq$ `y` and `Bi = x` for numerical column and categorical column, respectively. Obviously, if there is no difference between sampler qualities, then there is no need to decide which sampler to select. Thus, we generated two groups of queries based on the quality gap, named Small Gap Query and Large Gap Query, where each group has 50 queries. For Small Gap Query and Large Gap Query, the ratio of the quality of the second best sampler and the best sampler are smaller than 1.5, and larger than 1.5, respectively.

Error Metric. We used relative error to measure estimation quality. Suppose the true query result is q and the estimated result is $\hat{q}$, then the error is computed as $|\frac{\hat{q}-q}{q}|$. To reduce the randomness of estimation, we run each query over 5 different samples and compute the average error. Given a set of queries, we reported their 90th percentile average error. If the error is 2%, it means that 90% of queries have an average error (of 5 runs) smaller than 2%.

Implementation and Settings. We implemented SamComb in Java. The budget was set to 144,000 by default, which was 5% of *Loan* and 2.5% of *TPCS*, respectively. The experiments


(a) TPCS, Small Gap


(b) TPCS, Large Gap


(c) Loan, Small Gap

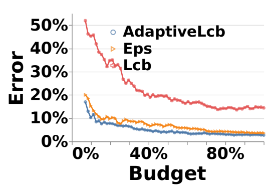
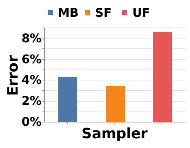
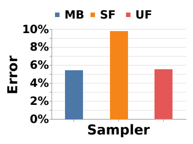
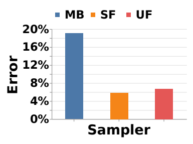
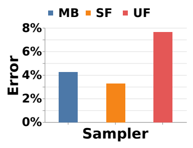

(d) Loan, Large Gap

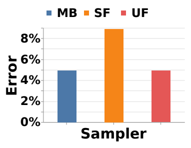
Figure 4.3: ϵ-greedy vs LCB vs AdaptiveLCB


(a) TPCS, U


(b) TPCS, S


(c) TPCS, M


(d) Loan, U


(e) Loan, S

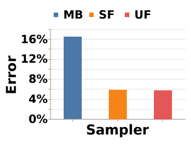

(f) Loan, M

Figure 4.4: Justification for Sampler Combination

were conducted on a MacBook Pro with an Intel Core i5 2.3GHz, 16GB RAM, and 250GB SSD.

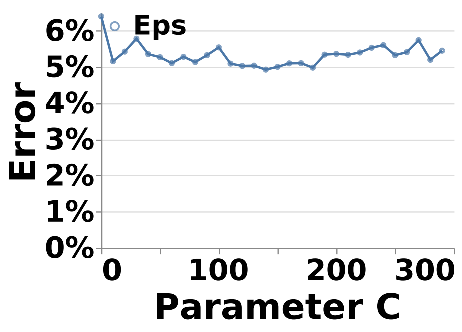
4.7.2 Evaluation of Our Approach

In this section, we evaluate SamComb under different settings. The goal is to find the best setting for SamComb. Note that ϵ-greedy has a parameter C but LCB does not. Thus, we first find the best parameter for ϵ-greedy, and then compare ϵ-greedy with two LCB-based approaches.

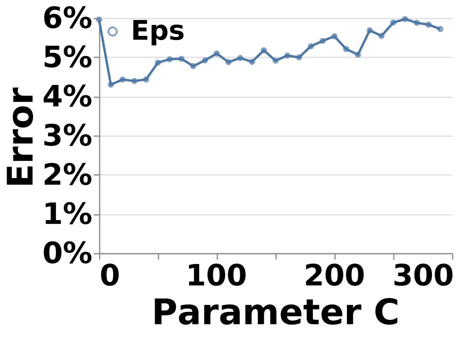
Parameter Selection for ϵ-greedy. Recall that $\epsilon_t = \min\{1, \frac{ck}{d^2t}\}$, where c and d are user given parameters [49]. For simplicity, let $C = \frac{ck}{d^2}$. As C increases, ϵ-greedy does more and more explorations.

We vary the parameter C to see how it affects the performance. The result is shown in Figure 4.5. We can observe that a small C usually works well. There are two reasons. First, if the sampler quality is very different from each other, then it is easy to distinguish them using a few explorations, thus a small C is preferred. Second, if the sampler quality is close to each other, choosing any sampler will not affect the performance much, thus a small C is also good. Based on this observation, we choose $C = 10$ for ϵ-greedy by default.

Comparing ϵ-greedy, LCB, and AdaptiveLCB. We compare ϵ-greedy, LCB, and AdaptiveLCB, aiming to find the best budget allocation strategy for SamComb. For a fair comparison, we randomly set the parameter for ϵ-greedy. The comparison results on the two datasets are shown in Figure 4.3. We have three observations. Firstly, LCB performed much worse than ϵ-greedy and AdaptiveLCB. This is because that LCB had a large bound and needed more sample to reduce the bound. It also shows that AdaptiveLCB got a good bound to automatically balance the exploration and exploitation trade-off. Secondly, there is a larger gap

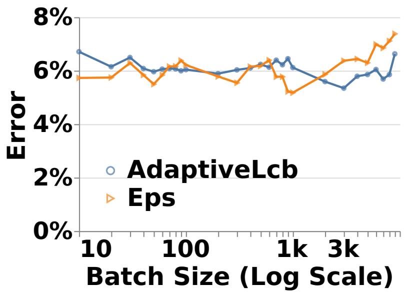


(a) TPCS

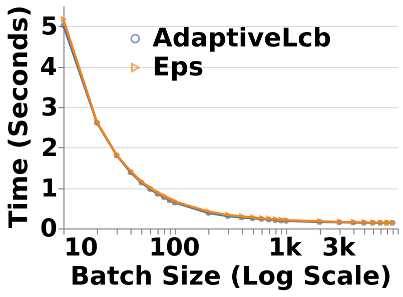


(b) Loan

Figure 4.5: Parameter Selection for ϵ-greedy



(a) Error



(b) Latency

Figure 4.6: Varying Batch Size

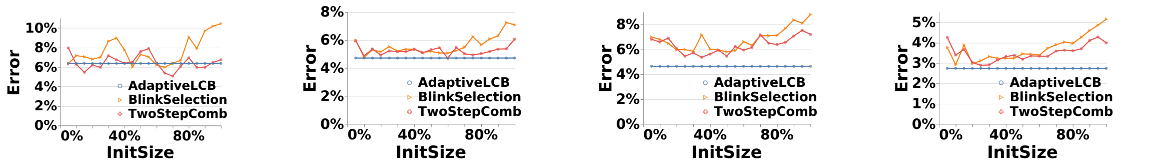
between different strategies when handling Large Gap queries than Small Gap queries. The underlying reason is that when samplers have similar qualities, the difference of selecting a different sampler is not big. Thirdly, AdaptiveLCB and ϵ-greedy had a similar performance and there is no clear winner. In TPCS, ϵ-greedy performs better in the beginning then AdaptiveLCB outperforms as budget increases. In Loan, AdaptiveLCB is slightly better.

In summary, AdaptiveLCB and ϵ-greedy outperformed LCB. Since AdaptiveLCB does not need to tune the parameter, AdaptiveLCB is chosen as the default strategy.

Varying Batch Size. SamComb draws a batch of tuples from a sampler in each iteration. In this experiment, we vary the batch size from 10 to 10000, and evaluate its impact on error and latency. The result is shown in Figure 4.6.

From Figure 4.6a, we can observe that the error is relative stable when the batch size is not very big. Although a small batch may be less accurate for estimating the sampler quality in each iteration, it also leads to more iterations, which is good for allocating more budgets to the best sampler. As a result, the error is relative stable.

Figure 4.6b shows that the latency could be very high for a small batch size, but it will keep stable when the batch size is above a threshold. This is because SamComb issues a query for each batch, and a small batch size will lead to many batches, causing a big



(a) TPCS, Small Gap (b) TPCS, Large Gap (c) Loan, Small Gap (d) Loan, Large Gap

Figure 4.7: Comparing SamComb, TwoStepComb and BlinkSelection

overhead. When the number of bathes is small, the overhead is negligible comparing to the query processing time, thus the latency becomes stable.

From this experiment, we can conclude that a poor setting of batch size could affects the latency a lot, while the impact on error happened slowly. Hence, we could choose the smallest batch size when increasing batch size does not further decrease the latency.

4.7.3 Comparison of Combination Approaches

In this section, we first justify the need for sampler combination and then compare different combination approaches.

One Size Does Not Fit All. We test the performance of the 100 queries with each individual sampler, and split the queries into three groups, named U, S and M group. In the U, S, and M group, the uniform sampler, the stratified sampler, and the measure-biased sampler performed worse than the other two samplers, respectively. We plot the performance for different samplers over the three groups of queries, as shown in Figure 4.4.

We can see that there is no single sampler that performs well in all cases. For example, on the Loan dataset, the stratified sampler performs much better than the other two samplers in the U group. However, it is much worse than the others in the S group. A similar observation can also be derived from the TPCS dataset. These results validate that one size does not fit all for sample-based estimation, and there is a strong need to combine multiple samplers.

Comparing with Best Sampler & Worst Sampler. For each query, there is a best sampler and a worst sampler, which is unknown unless scanning the full data. In this experiment, we comparing SamComb with two approaches: i) BestAlways: always choosing the best sampler. ii) WorstAlways: always choosing the worst sampler. The purpose of this experiment is to understand how far the estimation of SamComb is close to BestAlways and WorstAlways. The result is shown in Figure 4.8d.

Figure 4.8d shows that the error of SamComb is close to BestAlways and is much more accurate than WorstAlways. This is because SamComb combines multiple samplers and allocates more budgets to the best sampler. We further justify this point by investigating the allocated tuples of SamComb to the best sampler and the worst sampler for each query. It turns out that there are 73% queries where the best sampler is allocated more than 90% of the budgets, while 95% queries where the worst sampler is allocated less than 10% budget. This result further proves that SamComb successfully allocates most budgets to the best sampler.

Bandit-based vs Heuristic-based. We compare our bandit-based approach with two heuristic approaches.

TwoStepComb allocates the budget in two step: it first allocates an initial budget to each sampler and estimates their qualities, and then it allocates all the remaining budget to the empirical best sampler, which is similar to the explore-first approach [156]. Finally, it combines the estimation from multiple samples.

BlinkSelection is the sample selection technique used in BlinkDB [47]. It builds the error-latency profile for each sample and chooses the one satisfying the error or latency threshold. In our scenario, the latency is proportional to sample size (since we sequentially scan the data) and the relationship of error and sample size is clear. That is, we only need to estimate the sampler quality, then the error-budget profile can be built. Hence, BlinkSelection is actually the same as TwoStepComb without the sample combination phase.

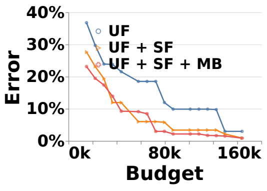
The performance of TwoStepComb and BlinkSelection depends on the initial budget size. We varied this parameter in TwoStepComb and BlinkSelection, and compared them with SamComb. Figure 4.7 shows the result. We see that SamComb outperformed TwoStepComb and BlinkSelection. This is because that SamComb allocated the budget adaptively, while TwoStepComb and BlinkSelection only used the initial estimation to decide the allocation of the remaining budget.

One major issue of TwoStepComb and BlinkSelection is that their performance is sensitive to the initial size. As shown in Figure 4.7, the performance of TwoStepComb and BlinkSelection first increased then decreased. This is because that at the beginning, TwoStepComb and BlinkSelection did not estimate the sampler quality accurately and chose a bad sampler as the empirical best sampler. With a larger initial budget, the quality estimation became more accurate and the probability of choosing the best sampler became larger. Hence, the performance was improved. However, as the initial size became larger and larger, the remaining budget became smaller and smaller. As a result, the performance of TwoStepComb and BlinkSelection decreased and it is more like equal allocation.

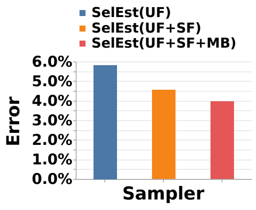
The result also indicates that the initial size is hard to tune. A good setting of initial size varies from query to query, and data to data. For example, a good initial size is around 40% and 20% of the total budget in Figure 4.7c and Figure 4.7d, respectively.

4.7.4 Evaluation in Various Settings

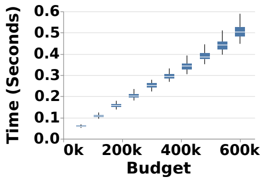
Support Other Aggregation Functions. We evaluate the performance of SamComb in supporting other aggregate functions. We pick up `MAX` because it is a challenging one for sample-based estimation. We used the same query workload but replaced the aggregate function with `MAX`. We calculated the rank of the estimated max value and get its relative



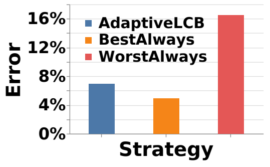
(a) Max Query



(b) Sel. Estimation



(c) Tuning Budget



(d) Comparing with BestAlways and WorstAlways

Figure 4.8: Evaluation in various settings (TPCS)

rank error based on $rank\_error = 1 - \frac{rank(estimate)}{N}$, where N is the population size. The result is shown in Figure 4.8a. We can see that SamComb can successfully combine multiple samplers and return a much more accurate answer to MAX queries compared to using a uniform sampler only. This is a promising result since it shows that combining multiple samplers enables sample-based estimation to handle difficult aggregation functions more accurately. We defer an extensive study of this direction to future work.

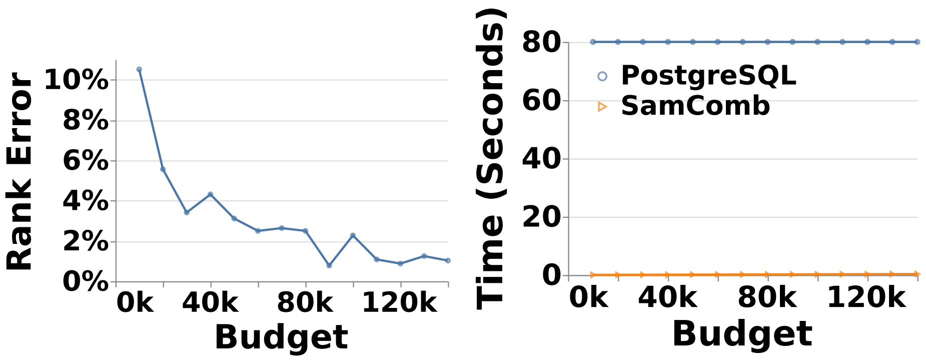
Selectivity Estimation. Selectivity estimation aims to estimate the percentage of the tuples that satisfy a predicate. It is an essential step in query optimization. Sampling is a common approach for solving this problem in existing database systems [140].

We evaluate selectivity estimation when multiple samplers are available. Selectivity estimation can be expressed using COUNT queries. Thus we replace the aggregation function in our queries from SUM to COUNT. The result is shown in Figure 4.8b. We can see that adding more samplers improves estimation accuracy. For example, adding the stratified sampler reduces the estimation error of the uniform sampler by 22%. We also notice that adding the measure-biased sampler improve less. This is because the measure-biased sampler is designed to sample more from the tuples with large measure values. However, in the selectivity estimation scenario, the queries are COUNT rather than SUM. Our SamComb framework automatically figured this out without wrongly selecting the measure-biased sampler to hurt the performance.

Tuning Budget. The budget size is a parameter used by AQP system to make the trade-off between the latency and error. Sometimes users may have a error or latency requirement, and want to tune the budget. This can be achieved by modeling the relationship of budget-error or budget-latency (e.g., the Error Latency Profiles in BlinkDB). We varied the budget and measured the latency of each query. Figure 4.8c shows the latency distribution of all queries under different budgets. We can find a linear relationship between budget and latency among different queries. This is because the query time is dominated by IO and the sample is scanned sequentially, thus the time is (approximately) proportional to the number of scanned tuples. It demonstrates that we can build a budget-latency profile to tune the budget based on the latency requirement.

Table 4.1: Evaluation of maintenance cost (10% new data)

	Sample Creation	**Sample Maintenance**
Uniform	29.33 secs	2.64 secs
Measure-biased	29.81 secs	2.92 secs
Stratified	29.09 secs	2.63 secs



(a) Relative Rank Error (b) Time (Seconds)

Figure 4.9: Performance of 90-Percentile Queries

Data Update. We tested the overhead of SamComb for data update (insert). We used the *TPCS* 1G as the original data, and took its 10% sample as the data to be inserted. The sample size is 1% of the original data for each sampler. We tested the time of creating an initial sample and the time of incrementally maintaining it. The result is shown in Table 4.1. We can see that the maintenance cost is relatively small. For example, the cost of maintaining a stratified sample is only $\frac{2.63}{29.09} = 9\%$ of the total sample creation time. This is because that incremental maintenance only needs to scan the stale sample and the delta table rather than the whole data.

4.7.5 End-to-end Performance

In this section, we conducted experiments on a TPCS data with scale 10 to show the end-to-end performance of SamComb.

Compare with VerdictDB and PostgreSQL. We compare the performance of SamComb with VerdictDB and PostgreSQL. The budget was set as 1% of the full data for SamComb and VerdictDB. The PostgreSQL is the approach that directly issue the query in PostgreSQL over the full data. The SamComb is built on top of PostgreSQL and each sampler is stored as a table in PostgreSQL. In each iteration, a query with predicate `row_id BETWEEN a AND b` is issued to get the statistics in the batch whose tuple id is between `a` and `b`. Then, the statistics are used to compute the estimated sampler quality.

Table 4.2: End-to-end performance comparison (Budget = 1%)

	Query Error	**Response Time**
VerdictDB	2.67%	0.36 secs
SamComb	1.89%	0.44 secs
PostgreSQL	0	32.93 secs

The result is shown in Table 4.2. From Table 4.2, we can see that under the same budget, the latency of SamComb is close to VerdictDB (but slightly slower). This is because they both scan samples sequentially at query time. It also shows that the overhead of SamComb is small. The reason is that the statistics used by SamComb to select a sampler can be computed incrementally. Furthermore, comparing to regular execution in PostgreSQL, SamComb can be round 80 times faster, since it only scans a small sample rather than the full data.

Percentile Query. In this experiment, we test the performance of SamComb for answering percentile queries. We choose 90th percentile, since the median percentile can handle by uniform sample well and extreme percentile is more challenging for AQP system.

We vary the budget from 10k to 120k, and evaluate the relative rank error and time of SamComb. The result is shown in Figure 4.9. Figure 4.9a shows that the estimation quality of SamComb for extreme percentile is improved as more budgets are allocated. This is because SamComb combines multiple samplers, and some samplers could draw tuples that are important to the extreme percentile with a higher probability. From Figure 4.9b, we can observe that the latency of SamComb scaled approximately linearly when the budget is not very big. This is because the main cost came from the IO scan of samples, rather than sorting the elements. Comparing to PostgreSQL whose latency is 80 seconds, SamComb can be 80 times faster within an error 2%.

4.8 Related Work

We review the related work on sample-based estimation, which can be divided into single-sample based and multiple-sample based.

Single Sample. Sample-based estimation has been extensively studied in both statistics [154] and databases [126, 109]. To estimate a population parameter, a simple approach is to draw a uniform sample and then use it to estimate the parameter. To improve the performance, various sampling techniques have been proposed. E.g., Sample+Seek [78] applies measure-biased sampling to improve uniform sampling. START [64] constructs an optimal stratified sample based on a given query workload. Congressional sampling [43] constructs a biased sample optimized for a set of group-by queries. Correlated sampling [162] constructs correlated samples based on the join key column. The main idea of these techniques is to increase the sampling probability for the tuples that are important to the result. There is another work also leverages MAB for sampling [68]. Different from us, it focuses on efficiently drawing a sample from a discrete random variable with a high degree of dependency.

Multiple Samples. There are some efforts that leverage multiple samples to improve estimation accuracy. E.g., BlinkDB [47] pre-computes multiple stratified samples and selects the best one using error-latency profile. Small group sampling [50] constructs multiple small group samples and selects samples based on group-by columns. VerdictDB [131] constructs different types of samples, and selects a sample using a heuristic cost model. Quickr [103, 102] applies heuristic rules to select various samplers in the sample plan and creates samples on the fly. SPEAr [104] targets at approximate stream processing scenario and selects uniform/stratified sample using heuristic rules. Bao [118] applies MAB to pick the appropriate hint for cardinality estimation. There is other related work [161] that combines samples from different distributions for Monte Carlo rendering. To the best of our knowledge, we are the first to study how to *dynamically* select and combine samplers. We show that this novel problem can be modeled as a MAB problem, and our solution balances the trade-off between exploration and exploitation in a principal way with theoretical guarantee.

4.9 Conclusion

In this work, we proposed a novel bandit-based framework, named SamComb, which combines multiple samplers to improve the quality of sample-based estimation. We formally defined the sampler combination problem and justified why it can be modeled as a multi-armed bandit problem. Our framework consists of three phases: (i) initialization phase, (ii) allocation phase, and (iii) combination phase. For the allocation phase, we proposed two strategies based on the well-known approaches in MAB, i.e., LCB and ϵ-greedy. We proved that they both allocates at most $\mathcal{O}(\ln n)$ budget to each sub-optimal sampler. For the combination phase, we proposed two weight allocation strategies to combine estimators, and proved that SamComb under the proportional-to-size allocation is asymptotically optimal. We extensively evaluated our approaches on both synthetic and real world datasets. The results showed that i) SamComb using the bandit-based approach (AdaptiveLCB) achieved higher performance than heuristic-based approaches (Random, TwoStepComb and BlinkSelection); ii) SamComb helped the reduce estimation error at the same sample budget (query latency).

Part II

Accelerating Human Analytics by Task-Centric API Design

Chapter 5

DataPrep.EDA: Task-Centric Exploratory Data Analysis in Python

This chapter presents DataPrep.EDA. It focuses on the exploratory data analysis (EDA) scenario and accelerates human analytics by task-centric API design. DataPrep.EDA allows data scientists to declaratively specify a wide range of EDA tasks in different granularity with a single function call. In this way, they can pay more attention to deciding what task to perform and leave the implementation details to the system.

5.1 Motivation

Python has grown to be one of the most popular programming languages in the world [36] and is widely adopted in the data science community. For example, the Python data science ecosystem, called PyData, is used by universities and online learning platforms to teach data science essentials [37, 19, 12, 27]. The ecosystem contains a wide range of tools such as Pandas for data manipulation and analysis, Matplotlib for data visualization, and Scikit-learn for machine learning, all aimed towards simplifying different stages of the data science pipeline.

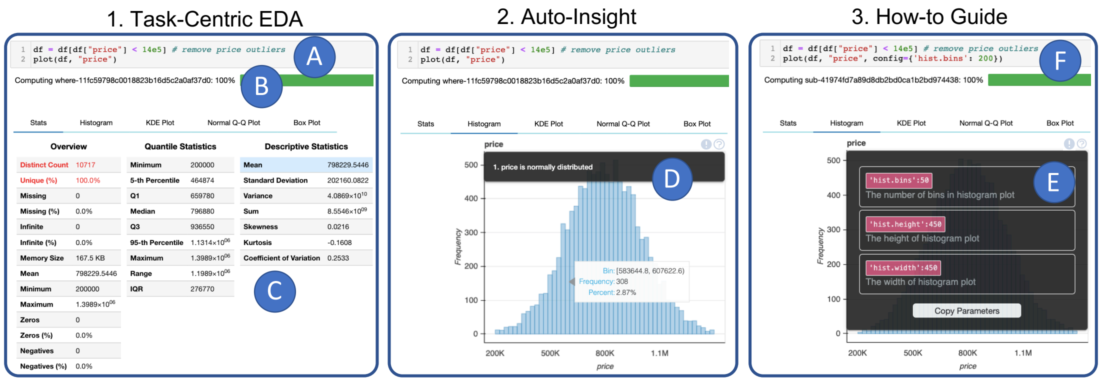
In this work we focus on one part of the pipeline, exploratory data analysis (EDA) for statistical modeling, the process of understanding data through data manipulation and visualization. It is an essential step in every data science project[173]. For statistical modeling, EDA often involves routine tasks such as understanding a single variable (univariate analysis), understanding the relationship between two random variables (bivariate analysis), and understanding the impact of missing values (missing value analysis).

Currently, there are two EDA solutions in Python. Each of them provide APIs in different granularity and have different drawbacks.

Pandas+Plotting. The first one is Pandas+Plotting, where Plotting represents a Python plotting library, such as Matplotlib [93], Seaborn [165], and Bokeh [55]. Fundamentally, plot-

Table 5.1: Comparison of EDA solutions in Python.

	Pandas+Plotting	Pandas-profiling	DataPrep.EDA
Easy to Use	×	✓	✓
Interactive Speed	✓	×	✓
Easy to Customize	✓	×	✓



```
df = df[df["price"] < 14e5] # remove price outliers
plot(df, "price")
```

Computing where-11fc59798c0018823b16d5c2a0af37d0: 100%

Overview	
Distinct Count	10717
Unique (%)	100.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	167.5 KB
Mean	798229.5446
Minimum	200000
Maximum	1.3989×10^{06}
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

Quantile Statistics	
Minimum	200000
5-th Percentile	464874
Q1	659780
Median	796880
Q3	936550
95-th Percentile	1.1314×10^{06}
Maximum	1.3989×10^{06}
Range	1.1989×10^{06}
IQR	276770

Descriptive Statistics	
Mean	798229.5446
Standard Deviation	202160.0822
Variance	4.0869×10^{10}
Sum	8.5546×10^{09}
Skewness	0.0216
Kurtosis	-0.1608
Coefficient of Variation	0.2533

```
df = df[df["price"] < 14e5] # remove price outliers
plot(df, "price")
```

Computing where-11fc59798c0018823b16d5c2a0af37d0: 100%

1. price is normally distributed

Bin: [583644.8, 607622.6)
Frequency: 308
Percent: 2.87%

```
df = df[df["price"] < 14e5] # remove price outliers
plot(df, "price", config={'hist.bins': 200})
```

Computing sub-41974fd7a89d8db2bd0ca1b2bd974438: 100%

'hist.bins':50
The number of bins in histogram plot

'hist.height':450
The height of histogram plot

'hist.width':450
The width of histogram plot

Figure 5.1: The front-end of DataPrep.EDA

ting libraries are *not* designed for EDA but for plotting. Their APIs are at a very low level, hence they are not easy to use: To complete an EDA task, a data scientist needs to think about what plots to create, then using Pandas to manipulate the data so that it can be fed into a plotting library to create these plots. Often there is a gap between an EDA task and the available plots – a data scientist must write lengthy and repetitive code to bridge the gap.

Pandas-profiling. The second one is Pandas-profiling [56]. It provides a very high level API and allows a data scientist to generate a comprehensive profile report. The report has five main sections: `Overview`, `Variables`, `Interactions`, `Correlation`, and `Missing Values`. Its general utility makes it the most popular EDA library in Python.

While Pandas-profiling is effective for one-time profiling, it suffers from two limitations for EDA due to its high-level API design: (i) Firstly, it does not achieve interactive speed since generating a profile report often takes a long time. This is suffering as EDA is an *iterative* process. Furthermore, the report shows information for all columns, potentially misdirecting the user and adding processing time. (ii) Secondly, it is not easy to customize a profile report. In a profile report, the plots are automatically generated thus it is very likely that the user wants to fine tune the parameters of each plot (e.g., the number of bins in a histogram). There could be hundreds of parameters associated with a profile report. It is not easy for users to figure out what they can customize and how to customize to meet their needs.

Table 5.1 summarizes the drawbacks of Pandas+Plotting and Pandas-profiling. The key challenge is how to overcome the limitations of existing tools and design a new EDA system that can achieve three design goals: easy to use, with interactive speed, and easy to customize. To address this challenge, we build DataPrep.EDA, a novel task-centric EDA system in Python. We identify a list of common EDA tasks for statistical modeling, mapping each task to a single function call through careful API design. As a result of this task-oriented approach, DataPrep.EDA affords many more *fine-grained* tasks such as univariate analysis and correlation analysis. Figure 5.1-1 illustrates how our example user might use DataPrep.EDA to do a univariate analysis task after removing their outliers. The analyst calls the `plot`(df, "price") in DataPrep.EDA, where df is a dataframe and "price" is the column name. DataPrep.EDA detects `price` as a numerical variable and automatically generates suitable statistics (e.g., max, mean, quantile) and plots (e.g., histogram, box plot), which help the user gain a deeper understanding of the `price` column quickly and effectively.

With the task-centric approach, DataPrep.EDA is able to achieve all three design goals: (i) *Easy to Use.* Since each EDA task is directly mapped to a single function call, users only need to think about what tasks to work on rather than what to plot and how to plot. To further improve usability, we design an *auto-insight* component to automatically highlight possible interesting patterns in visualizations. (ii) *Interactive Speed.* Different from Pandas-profiling, DataPrep.EDA supports fine-grained tasks thus it can avoid unnecessary computation on irrelevant information. To further improve the speed, we carefully design our data processing pipeline based on Dask, a scalable computing framework in Python. (iii) *Easy to Customize.* With the task-centric API design, the parameters are grouped by different EDA tasks and each API only contains a small number of task-related parameters, making it much easier to customize. Besides, we implement a *how-to guide* component in DataPrep.EDA to further improve the customizability.

We conduct extensive experiments to compare DataPrep.EDA with Pandas-profiling. The performance results on 15 real-world datasets from Kaggle [22] show that i) DataPrep.EDA responded to a fine-grained EDA task in seconds while Pandas-profiling spent several orders of magnitude more time in creating a profile report on the same dataset; ii) if the task is to create a profile report, DataPrep.EDA was $4-20\times$ faster than Pandas-profiling. Through a user study we show that i) real world participants of varying skill levels completed 2.05 times more tasks on average with DataPrep.EDA than with Pandas-profiling; ii) DataPrep.EDA helped participants answering 2.20 times more correct answers.

The following summarizes our contributions:

- We explore the limitations of existing EDA solutions in Python and propose a task-centric framework to overcome them.
- We design a task-centric EDA API for statistical modeling, allowing to declaratively specify an EDA task in one function call.
- We identify three challenges to implement DataPrep.EDA, and propose effective solutions to enhance the scalability, usability, and customizability of the system.

EDA Task	Task-Centric API Design	Corresponding Stats/Plots
Overview	**plot**(df)	Dataset statistics, histogram or bar chart for each column
Univariate Analysis	**plot**(df, col_1)	(1) N ⟶ Column statistics, histogram, KDE plot, normal Q-Q plot, box plot (2) C ⟶ Column statistics, bar chart, pie chart, word cloud, word frequencies
Bivariate Analysis	**plot**(df, col_1, col_2)	(1) NN ⟶ Scatter plot, hexbin plot, binned box plot (2) NC or CN ⟶ Categorical box plot, multi-line chart (3) CC⟶ Nested bar chart, stacked bar chart, heat map
Correlation Analysis	**plot_correlation**(df)	Correlation matrix, computed with Pearson, Spearman, and KendallTau
	plot_correlation(df, col_1)	Correlation vector, computed with Pearson, Spearman, and KendallTau
	plot_correlation(df, col_1, col_2)	Scatter plot with a regression line
Missing Value Analysis	**plot_missing**(df)	Bar chart, missing spectrum plot, nullity correlation heatmap, dendrogram
	plot_missing(df, col_1)	Histogram or bar chart that shows the impact of the missing values in col_1 on all other columns
	plot_missing(df, col_1, col_2)	Histogram, PDF, CDF, and box plot that show the impact of the missing values from col_1 on col_2

Figure 5.2: A set of mapping rules between EDA tasks and corresponding stats/plots (N = Numerical, C = Categorical)

- We conduct extensive experiments to compare DataPrep.EDA with Pandas-profiling, the state-of-the-art EDA system in Python. The results show that DataPrep.EDA significantly outperforms Pandas-profiling in speed, effectiveness, and user preference.

5.2 Task-Centric EDA

In this section, we first introduce common EDA tasks for statistical modeling, and then describe our task-centric EDA API design.

5.2.1 Common EDA Tasks for Statistical Modeling

Inspired by the profile report generated by Pandas-profiling and existing work [136, 146, 54], we identify five common EDA tasks. We will use a running example to illustrate why they are needed in the process of statistical modeling.

Suppose a data scientist wants to build a regression model to predict house prices. The training data consists of four features (`size`, `year_built`, `city`, and `house_type`) and the target (`price`).

- **Overview**. At the beginning, the data scientist has no idea about what's inside the dataset, so she wants to get a quick overview of the entire dataset. This involves computing some basic statistics and creating some simple visualizations. For example, she may want to check the number of features, the data type of each feature (numerical or categorical), and create a histogram for each numerical feature and a bar chart for each categorical feature.

- **Correlation Analysis**. To select important features or identify redundant features, correlation analysis is commonly used. It computes a correlation matrix, where each cell in the matrix represents the correlation between two columns. A correlation matrix can show which features are highly correlated with the target and which two features are highly correlated with each other. For example, if the feature, `size`, is highly correlated with the target, `price`, then knowing `size` will reveal a lot of information about `price`, thus it is an important feature. If two features, `city` and `house_type`, are highly correlated, then one of the features is redundant and can be removed

- **Missing Value Analysis**. It is more common than not for a dataset to have missing values. The data scientist needs to create customized visualizations to understand missing values. For example, she may create a bar chart, which depicts the amount of missing values in each column, or a missing spectrum plot, which visualizes which rows has more missing values.
- **Univariate Analysis**. Univariate analysis aims to gain a deeper understanding of a single column. It creates various statistics and visualizations of that column. For example, to deeply understand the feature `year_built`, the data scientist may want to compute the min, max, distinct count, median, variance of `year_built`, and create a box plot to examine outliers, a normal Q-Q plot to compare its distribution with the normal distribution.
- **Bivariate Analysis**. Bivariate analysis is to understand the relationship between two columns (e.g., a feature and the target). There are many visualizations to facilitate the understanding. For example, to understand the relationship between `year_built` and `price`, she may want to create a scatter plot to check whether they have a linear relationship, and a hexbin plot to check the distribution of `price` in different year ranges.

There are certainly other EDA tasks used for statistical modeling, however we have opted to focus on the main tasks systems such as Pandas-profiling commonly present in their reports. This allows us to make a fair comparison between our system design approaches. In the future we intend to address more tasks, such as time-series analysis and multi-variate analysis (more than two variables).

5.2.2 DataPrep.EDA's Task-Centric API Design

The goal of our API design is to enable the user to trigger an EDA task through a single function call. We consider simplicity and consistency as the principle of API design. The simple and consistent API makes our system more accessible in practice [57]. However, it is challenging to design simple and consistent APIs for a variety of EDA tasks. Our key observation is that the EDA tasks for statistical modeling tend to follow a similar pattern [166]: start with an overview analysis and then dive into detailed analysis. Hence, we design the API in the following form:

$$\textbf{plot\_}\textbf{\textit{tasktype}}(\mathsf{df}, \mathsf{col\_list}, \mathsf{config}),$$

where plot_*tasktype* is the function name, tasktype is a concise description of the task, the first argument is a DataFrame, the second argument is a list of column names, and the third argument is a dictionary of configuration parameters. If column names are not specified, the task will be performed on all the columns in the DataFrame (overview analysis); otherwise, it will be performed on the specified column(s) (detailed analysis). This design makes the API extensible, i.e., it is easy to add an API for a new task.

Following this pattern, we design three functions in DataPrep.EDA to support the five EDA tasks:

plot. We use the `plot`$(\cdot)$ function with different arguments to represent the overview task, the univariate analysis task, and the bivariate analysis task, respectively. To understand how to perform EDA effectively with this function, the following gives the syntax of the function call with the intent of the data scientist:

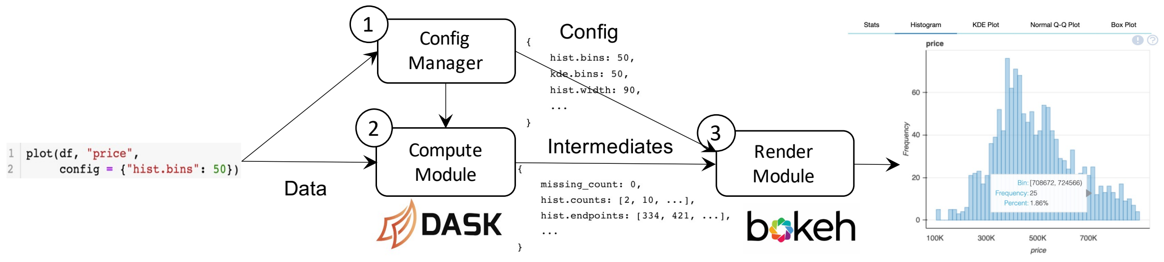
- plot(df): "I want an overview of the dataset"
- plot(df, $\texttt{col}_1$): "I want to understand $\texttt{col}_1$"
- plot(df, $\texttt{col}_1$, $\texttt{col}_2$): "I want to understand the relationship between $\texttt{col}_1$ and $\texttt{col}_2$"

plot_correlation. The `plot_correlation`$(\cdot)$ function triggers the correlation analysis task. The user can get more detailed correlation analysis results by calling `plot_correlation`(df, col_1) or `plot_correlation`(df, col_1, col_2).

- plot_correlation(df): "I want an overview of the correlation analysis result of the dataset"
- plot_correlation(df, $\texttt{col}_1$): "I want to understand the correlation between $\texttt{col}_1$ and the other columns"
- plot_correlation(df, $\texttt{col}_1$, $\texttt{col}_2$): "I want to understand the correlation between $\texttt{col}_1$ and $\texttt{col}_2$"

plot_missing. The `plot_missing`$(\cdot)$ function triggers the missing value analysis task. Similar to `plot_correlation`$(\cdot)$, the user can call `plot_missing`(df, col_1) or `plot_missing`(df, col_1, col_2) to get more detailed analysis results.

- plot_missing(df): "I want an overview of the missing value analysis result of the dataset"
- plot_missing(df, $\texttt{col}_1$): "I want to understand the impact of removing the missing values from $\texttt{col}_1$ on other columns"
- plot_missing(df, $\texttt{col}_1$, $\texttt{col}_2$): "I want to understand the impact of removing the missing values from $\texttt{col}_1$ on $\texttt{col}_2$"



```
plot(df, "price",
     config = {"hist.bins": 50})
```

```
{
  hist.bins: 50,
  kde.bins: 50,
  hist.width: 90,
  ...
}
```

```
{
  missing_count: 0,
  hist.counts: [2, 10, ...],
  hist.endpoints: [334, 421, ...],
  ...
}
```

Figure 5.3: The DataPrep.EDA system architecture

The key observation that makes task-centric EDA possible is that there are nearly universal kinds of stats or plots that analysts employ in a given EDA task. For example, if the user wants to perform univariate analysis on a numerical column, she will create a histogram to check the distribution, a box plot to check the outliers, a normal Q-Q plot to compare with the normal distribution, etc. Based on this observation, we pre-define a set of mapping rules as shown in Figure 5.2, where each rule defines what stats/plots to create for each EDA task. Once a function, e.g., `plot(df,"price")`, is called, DataPrep.EDA first detects the data type of `price`, which is numerical. Based on the second row in Figure 5.2, since $\text{col}_1 = \text{N}$, DataPrep.EDA will automatically generate the column statistics, histogram, KDE plot, normal Q-Q plot, and box plot of `price`.

The mapping rules are selected from existing literature and open-source tools in the statistics and machine learning community. For univariate, bivariate, and correlation analysis, we refer to the data-to-viz project [16], 'Exploratory Graphs' section in [136] and Section 4 in [146]. For overview analysis and missing value analysis, the mapping rules are derived from the Pandas-profiling library and the Missingno library [54], respectively. DataPrep.EDA also leverages the open-source community to keep adding and improving its rules. For example, one user has created an issue in our GitHub repository to suggest adding violin plots to the `plot(df,x)` function. As DataPrep.EDA is being used by more users, we expect to see more suggestions like this in the future.

5.3 System Architecture

This section describes DataPrep.EDA's front-end user experience and introduces the back-end system architecture.

5.3.1 Front-end User Experience

To demonstrate DataPrep.EDA, we will continue our house price prediction example, using DataPrep.EDA to assist in removing outliers from the `price` variable, assessing the resulting distribution, and determining how to further customize the analysis. Figure 5.1 depicts the steps to perform this task using Pandas and DataPrep.EDA in a Jupyter notebook. Part **A** shows the required code: in line 1, the records with an outlying price value are removed (the threshold is \$1,400,000), and in line 2 the DataPrep.EDA function `plot` is called to analyze the filtered distribution of the variable `price`. Part Ⓑ shows the progress bar. Part Ⓒ shows the default output tab which consists of tables containing various statistics of the column's distribution. Note that each data visualization is output in a separate panel, and tabs are used to navigate between panels.

- **Auto-Insight**. If an insight is discovered by DataPrep.EDA, a (❗) icon will be shown on the top right corner of the associated plot. Part Ⓓ shows an insight associated with the histogram plot: `price` is normally distributed.
- **How-to Guide**. A how-to guide will pop up after clicking a (❓) icon. As shown in Part Ⓔ, it contains the information about customizing the associated plot. In this example, the data scientist may want to create a histogram with more bins, so she can copy the code (`"hist.bins":50`) in the how-to guide, paste it as a parameter to the plot function, and increase the number of bins from 50 to 200 as shown in Part Ⓕ.

5.3.2 Back-end System Architecture

The DataPrep.EDA back-end is presented in Figure 5.3, consisting of three components: ① The Config Manager configures the system's parameters, ② the Compute module performs the computations on the data, and ③ the Render module creates the visualizations and layouts. The Config Manager is used to organize the user-defined parameters and set default parameters in order to avoid setting and passing many parameters through the Compute and Render modules. The separation of the Compute module and the Render module has two benefits: First, the computations can be distributed to multiple visualizations. For example, in `plot(df, col`$_1$`=N)` in Figure 5.2, the column statistics, normal Q-Q plot, and box plot all require quantiles of the distribution. Therefore, the quantiles are computed once and distributed appropriately to each visualization. Second, the intermediate computations (see Section 5.3.2) can be exposed to the user. This allows the user to create the visualizations with her desired plotting library.

Config Manager

The Config Manager (① in Figure 5.3) sets values for all configurable parameters in DataPrep.EDA, and stores them in a data structure, called the `config`, which is passed through the rest of the system. Many components of DataPrep.EDA are configurable including which visualizations to produce, the insight thresholds (see Section 5.3.2), and visualization customizations such as the size of the figures. In Figure 5.3, the `plot` function is called with the user specification `bins=50`; the Config Manager sets each bin parameter to have a value of 50, and default values are set for parameters not specified by the user. The `config` is then passed to the Compute and Render modules and referenced when needed.

Compute module

The Compute module takes the data and `config` as input, and computes the `intermediates`. The `intermediates` are the results of all the computations on the data that are required to generate the visualizations for the EDA task. Figure 5.3 shows example `intermediates`. The first element is the count of missing values which is shown in the `Stats` tab, and the next two elements are the counts and bin endpoints of the histogram. Such statistics are ready to be fed into a visualization.

Insights are calculated in the Compute module. A data fact is classified as an insight if its value is above a threshold (each insight has its own, user-definable threshold). For example, in Figure 5.1 (Part Ⓑ), the distinct value count is high, so the entry in the table is highlighted red to alert the user about this insight. DataPrep.EDA supports a variety of insights including data quality insights (e.g., missing, infinite values), distribution shape insights (e.g., uniformity, skewness) and whether two distributions are similar.

We developed two optimization techniques to increase performance. First, we share computations between multiple visualizations as described in the beginning of Section 5.3.2. Second, we leverage Dask to parallelize computations (see Section 5.4 for details).

Render module

The last system component is the Render module, which converts the `intermediates` into data visualizations. There is a plethora of Python plotting libraries (e.g., Matplotlib, Seaborn, and Bokeh), however, they provide limited or no support for customizing a plot's *layout*. A *layout* is the surrounding environment in which visualizations are organized and embedded. Our layouts need to consolidate many elements including charts, tables, insights, and how-to guides. To meet our needs, we use the library Bokeh to create the plots, and embed them in our own HTML/JS layout.

5.4 Implementation

In this section, we introduce the detailed implementation of DataPrep.EDA's Compute module. We first introduce the background of Dask and discuss why we choose Dask as the back-end engine. We then present our ideas for using Dask to optimize DataPrep.EDA.

5.4.1 Why Dask

Dask Background. Dask is an open source library providing scalable analytics in Python. It offers similar APIs and data structures with other popular Python libraries, such as NumPy, Pandas, and Scikit-Learn. Internally, it partitions data into chunks, and runs computations over chunks in parallel.

The computations in Dask are lazy. Dask will first construct a computational graph that expresses the relationship between tasks. Then, it optimizes the graph to reduce computations such as removing unnecessary operators. Finally, it executes the graph when an eager operation like `compute` is called.

Choice of Back-end Engine. We use Dask as the back-end engine of DataPrep.EDA for three reasons: (i) it is lightweight and fast in a single-node environment, (ii) it can scale to a distributed cluster, and (iii) it can optimize the computations required for multiple visualizations via lazy evaluation. We considered other engines like Spark variants [175, 41] (PySpark and Koalas) and Modin [137], but found that they were less suitable for DataPrep.EDA than Dask. Since Spark is designed for computations on very big data (TB to PB) in a large cluster, PySpark and Koalas are not lightweight like Dask and have a high scheduling overhead on a single node. For Modin, most of its operations are eager, so for each operation a separate computational graph is created. This approach does not optimize across operations, unlike Dask's approach. In Section 5.5.2, we further justify our choice to use Dask experimentally.

5.4.2 Performance Optimization

Given an EDA task, e.g., `plot_missing(df)`, we discuss how to efficiently compute its `intermediates` using Dask. We observe that there are many redundant computations between visualizations. For example, as shown in Figure 5.2, `plot_missing(df)` creates four visualizations (bar chart, missing spectrum plot, nullity correlation heatmap, and dendrogram). They share many computations, such as computing the number of rows, checking whether a cell is missing or not. To leverage Dask to remove redundant computations, we seek to express all the computations in a *single* computational graph. To implement this idea, we can make all the computations lazy and call an eager operation at the end. In this way, Dask will optimize the whole graph before actual computations happen.

However, there are several issues with this implementation. In the following, we will discuss them and propose our solutions.

Dask graph fails to build. The first issue is that the `rechunk` function in Dask cannot be directly incorporated into the big computational graph. This is because that its first argument, a Dask array, requires knowing the chunk size information, i.e., the size of each chunk in each dimension. If `rechunk` was put into our computational graph, an error would be raised since the chunk size information is unknown for a delayed Dask array.

Since `rechunk` is needed in multiple `plot` functions in DataPrep.EDA, we have to address this issue. One solution is to replace the `rechunk` function call in each `plot` function with the code written by the low-level Dask task graph API. However, this solution has a high engineering cost, which requires writing hundreds of lines of Dask code. It also has a high maintenance cost compared to using the Dask built-in `rechunk` function.

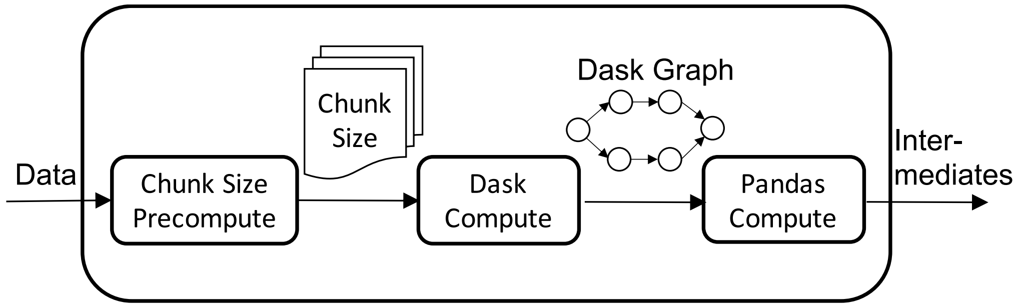


Figure 5.4: Data processing pipeline in the Compute module

We propose to add an additional stage before constructing the computational graph. In this stage, we precompute the chunk size information of the dataset and pass the precomputed chunk size to the Dask graph. In this way, the Dask graph can be constructed successfully by adding one line of code.

Dask is slow on tiny data. Although putting all possible operations in the graph can fully leverage Dask's optimizations, it also increases the overhead caused by scheduling. When data is large, the scheduling overhead is negligible compared to the computing overhead. However, when data is tiny, the scheduling may be the bottleneck and using Dask is less efficient than using Pandas.

For the nine `plot` functions in Figure 5.2, we observe that they all follow the same pattern: the computational graph takes as input a DataFrame (large data) and continuously reduces its size by aggregation, filtering, etc. Based on this observation, we separate the computational graph into two parts: *Dask Computation* and *Pandas Computation*. In the *Dask Computation*, the data is computed in Dask and the result is transformed into a Pandas DataFrame. In the *Pandas Computation*, it takes the DataFrame as input and does some further processing to generate the intermediate results, which will be used to create visualizations.

Currently, we heuristically determine the boundary between the two parts. For example, the computation for plot_correlation(df) is separated into two stages. In the first stage we use Dask to compute the correlation matrix from the user input and then in the second stage we use Pandas to transform and filter the correlation matrix. This is because for a dataset with n rows and m columns, it is usually the case that $n >> m$. As a result, it would be beneficial to let Pandas handle the correlation matrix, which has the size $m \times m$. Since we only need to handle nine `plot` functions, it is still manageable. We will investigate how to automatically separate the two parts in the future.

Putting everything together. Figure 5.4 shows the data processing pipeline in the Compute module of DataPrep.EDA. When data comes, DataPrep.EDA first precomputes the chunk size information using Dask. After that, it constructs the computational graph using Dask again with the precomputed chunk size filled. Then, it computes the graph. After that, it transforms the computed data into Pandas and finishes the Pandas computation. In the end, the `intermediates` are returned.

5.5 Experimental Evaluation

In this section, we conduct extensive experiments to evaluate the efficiency and the user experience of DataPrep.EDA.

5.5.1 Performance Evaluation

We first evaluate the performance of DataPrep.EDA by comparing its report functionality with Pandas-profiling. Afterwards, we conduct a self comparison, which evaluates each plot function with different variations, in order to gain a deep understanding of the performance. We used 15 different datasets varying in number of rows, number of columns and categorical/numerical column ratio, as listed in Table 5.2. The experiments were performed on an Ubuntu 16.04 Linux server with 64 GB memory and 8 Intel E7-4830 cores.

Comparing with Pandas-profiling. To test the general efficiency of DataPrep.EDA, we compared the end-to-end running time of generating reports over the 15 datasets in both tools. For Pandas-profiling, we first used `read_csv` from Pandas to read the data, then we created a report in Pandas-profiling with PhiK, Recoded and Cramer's V correlations disabled (since DataPrep.EDA does not implement these correlation types). For DataPrep.EDA, we used `read_csv` from Dask to read the dataset. Since it is a one-time task to create a profiling report, loading the data using the most suitable function for each tool is reasonable. Results are shown in Table 5.2. We observe that using DataPrep.EDA to generate reports is 4x $\sim$ 20x faster compared to Pandas-profiling in general. The acceleration mainly comes from the optimization to make the tasks into a single Dask graph so that they can be fully parallelized. We also observe that DataPrep.EDA usually gains more performance compared to Pandas-profiling on numerical data and data with fewer categorical columns (driving performance on credit, basketball, and diabetes).

Self-comparison. We analyzed the running time of DataPrep.EDA's functions to determine whether they can complete within a reasonable response time for interactivity. We ran `plot()`, `plot_correlation()`, and `plot_missing()` for each column in each dataset, and we ran the three functions for all unique pairs of columns in each dataset (limited to categorical columns with no more than 100 unique values for `plot(df, col`$_1$`, col`$_2$`)` so the resulting visualizations contain a reasonable amount of information). Figure 5.5 shows the percent of total tasks for each function that finish within 0.5, 1, 2 and 5 seconds. Note that dataset loading time is also included in the reported run times. The majority of tasks completed within 1 second for each function except `plot_missing(df, col`$_1$`)`. `plot_missing(df, col`$_1$`)` is computationally intensive because it computes two frequency distributions for each column (before and after dropping the missing values in column col$_1$).

Table 5.2: Comparing DataPrep.EDA with Pandas-profiling on 15 real-world data science datasets from Kaggle (**N** = Numerical, **C** = Categorical, PP=Pandas-profiling).

Dataset	**Size**	**#Rows**	**#Cols (N/C)**	**PP**	**DataPrep**	**Faster**
heart [17]	11KB	303	14 (14/0)	17.7s	2.0s	8.6×
diabetes [26]	23KB	768	9 (9/0)	28.3s	1.6s	17.7×
automobile [7]	26KB	205	26 (10/16)	38.2s	3.9s	9.8×
titanic [39]	64KB	891	12 (7/5)	17.8s	2.1s	8.5×
women [40]	500KB	8553	10 (5/5)	19.8s	2.3s	8.6×
credit [14]	2.7MB	30K	25 (25/0)	127.0s	6.1s	20.8×
solar [31]	2.8MB	33K	11 (7/4)	25.1s	2.7s	9.3×
suicide [33]	2.8MB	28K	12 (6/6)	20.6s	2.8s	7.4×
diamonds [15]	3MB	54K	11 (8/3)	28.2s	3.1s	9×
chess [11]	7.3MB	20K	16 (6/10)	23.6s	4.3s	5.5×
adult [5]	5.7MB	49K	15 (6/9)	23.2s	4.0s	5.8×
basketball [9]	9.2MB	53K	31 (21/10)	126.2s	9.9s	12.7×
conflicts [4]	13MB	34K	25 (10/15)	34.9s	8.6s	4×
rain [29]	13.5MB	142K	24 (17/7)	100.1s	11.6s	8.6×
hotel [18]	16MB	119K	32 (20/12)	83.2s	13s	6.4×

5.5.2 Experiments on Large Data

We conduct experiments to justify the choice of Dask (see Section 5.4.1) and evaluate the scalability of DataPrep.EDA. We use the bitcoin dataset [10] which contains 4.7 million rows and 8 columns.

Comparing Engines. We compare the time required for Dask, Modin, Koalas, and PySpark to compute the `intermediates` of `plot`(df). The results are shown in Figure 5.6(a). The reason why Dask is the fastest is explained in Section 5.4.1: Modin eagerly evaluates the computations and does not make full use of parallelization when computing multiple visualizations, and Koalas/PySpark have a high scheduling overhead in a single-node environment.

Varying Data Size. To evaluate the scalability of DataPrep.EDA, we compare the report functionality of DataPrep.EDA with Pandas-profiling and vary the data size from 10 million to 100 million rows. The data size is increased by repeated duplication. The results are shown in Figure 5.6(b). Both DataPrep.EDA and Pandas-profiling scale linearly, but DataPrep.EDA is around six times faster. This is because DataPrep.EDA leverages lazy evaluation to express all the computations in a single computational graph so that the computations can be fully parallelized by Dask.

Varying # of Nodes. To evaluate the performance of DataPrep.EDA in a cluster environment, we run the report functionality on a cluster and vary the number of nodes. The cluster consists of a maximum of 8 nodes, each with 64GB of memory and 16 2.0GHz E7-4830 CPUs dedicated to the Dask workers. There are also HDFS services running on the 8 nodes for data storage; the memory for these services is not shared with the Dask workers.

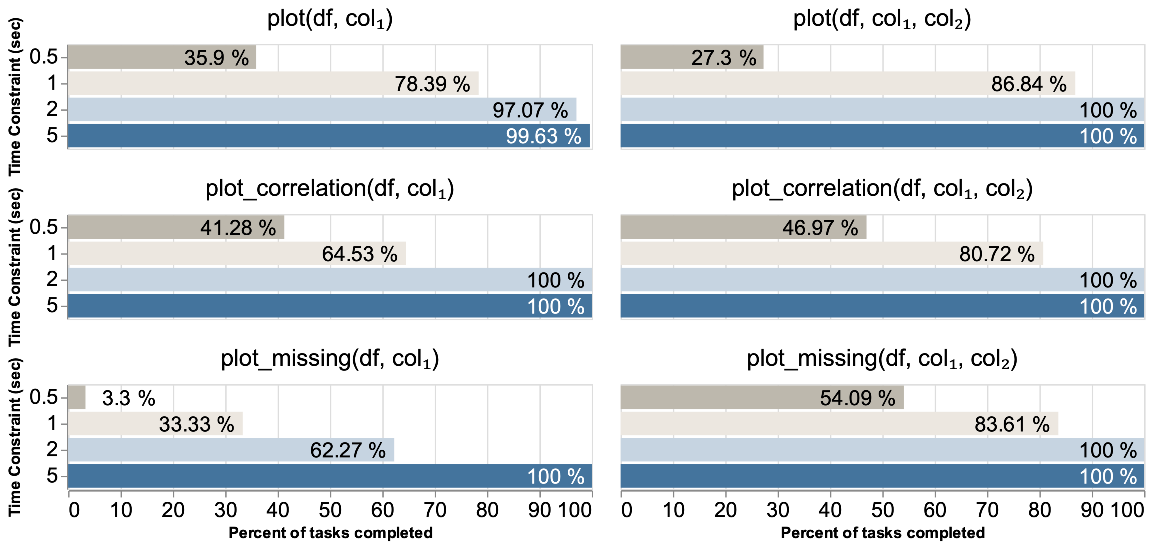


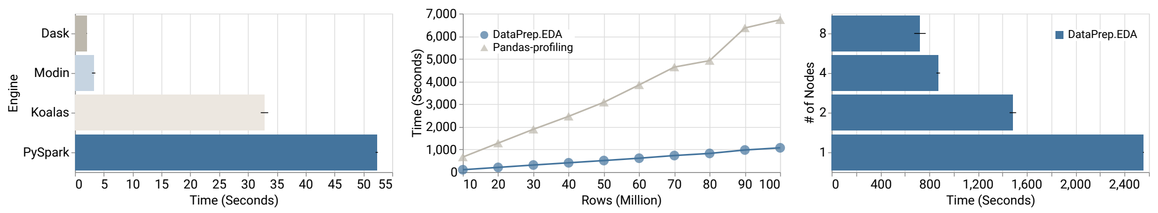
Figure 5.5: The percentage of tasks to finish within the given time constraint.

The data is fixed at 100 million rows and stored in HDFS. We do not compare with Pandas-profiling since it cannot run on a cluster. The result is shown in Figure 5.6(c). We can see that DataPrep.EDA is able to run on a cluster and achieves better performance as increasing the number of nodes. This is because adding more compute nodes can reduce the I/O cost of reading data from HDFS. It is worth noting that the 1 worker setting in Figure 5.6(c) is different from the single node setting in Figure 5.6(b) where the former needs to read data from HDFS while the latter reads data from a local disk. Therefore, the 1 worker setting took longer to process 100 million rows than the single node setting.

5.5.3 User Study

Finally, we conducted a user study to validate the usability of our tool and its utility in supporting analytics. We focus on two questions using Pandas-profiling as a comparison baseline: (1) For different groups of participants, how do they benefit from the task-centric features introduced by DataPrep.EDA, and (2) Does DataPrep.EDA reduce participants' false discovery rate. We hypothesize that the intuitive API in DataPrep.EDA will lead participants to complete more tasks in less time versus Pandas-profiling, and that the additional tools provided by DataPrep.EDA will help participants to reduce their false discovery rate.

Methodology. In our study, all recruited participants used both DataPrep.EDA and Pandas-profiling to complete two different sets of tasks in a 50 minute session with software logging: one tool is used for one set of tasks. Afterwards, they assessed both systems in surveys. Our study employs two datasets paired with task sets: (1) BirdStrike: 12 columns related to bird strike damage on airplanes. The dataset compiles approximately 220, 000 strike reports from 2, 050 USA airports and 310 foreign airports. (2) DelayedFlights: 14 columns related



(a) Comparing Engines (Single Node) (b) Varying Data Size (Single Node) (c) Varying # of Nodes

Figure 5.6: Experiments on the Bitcoin Dataset: (a) Comparing the running time of using different engines to compute visualizations in plot(df); (b) Comparing the running time of create_report(df) of DataPrep.EDA and Pandas-profiling by varying data size; (c) Evaluating the running time of create_report(df) of DataPrep.EDA by varying the number of nodes.

to the causes of flight cancellations or delays. The dataset is curated by the Department of Transportation of United States, and contains $5,819,079$ records of cancelled flights.

We followed a within-subjects design, with all participants making use of either tool to complete one set of tasks. We counterbalanced our allocation to account for all tool-dataset permutations and order effects. Within each task, participants finished five sequential tasks using one tool. As experience might influence how much an individual benefits from each tool, we recruited both skilled and novice analysts using a pre-screen concerning knowledge of python, data analysis, and the datasets in the study. To make sure that all participants had base knowledge of how to use both Pandas-profiling and DataPrep.EDA, we gave participants with two introductory videos, a cheat sheet, and API documentation.

Participants were asked to complete 5 tasks sequentially using the data analysis tool. The tasks are designed to evaluate different functions provided by DataPrep.EDA, which is similar to the design of existing work [52]. They cover a relatively wide spectrum of the kinds of tasks that are frequently encountered in EDA, including gathering descriptive multivariate statistics of one or multiple columns, identifying missing values, and finding correlations. Though datasets have their own specific task instructions, each of the respective items shares the same goal across both datasets. For example, the first task of both sessions asks participants to investigate data distribution over multiple columns.

In Task 1-3, participants use the provided tool to analyze the distribution of single or multiple columns. Participants conduct a univariate analysis in task 1 and a variate analysis in task 2. The task 3 asked the participant to examine distribution skewness. Task 4 examines missing values and their impact. Participants are expected to examine the distribution of missing values to come to a conclusion. Task 5 asks users to find columns with high correlation.

We use the fraction $\frac{\#correctanswers}{\#completedtasks}$ to show the *relative accuracy* of participants, since we noticed that a number of participants failed to finish tasks. Compared to traditional accuracy, relative accuracy therefore may better demonstrate positive discovery rate [60].

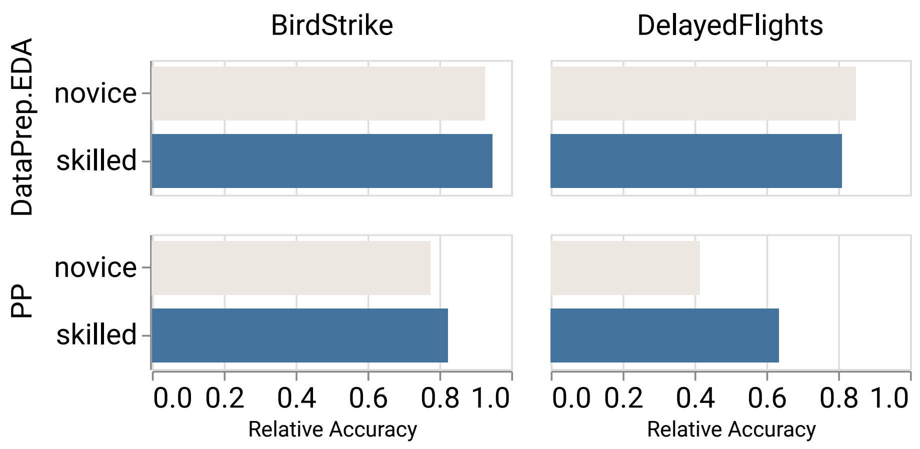


Figure 5.7: Relative Accuracy of DataPrep.EDA and Pandas-profiling across different skill levels of participants in dataset BirdStrike and DelayedFlights.

Examining Participants' Performance. We examine accuracy and completed tasks to evaluate system performance. The average number of completed tasks per participant using DataPrep.EDA (M:4.02, SD:1.21) was ***2.05*** times higher than that using Pandas-profiling (M:1.96, SD: 2.59, t(29)=5.26, $\rho < .00001$). When we compared skill levels, we could detect no difference (t(14)=.882998, $\rho < .441499$). Together, this suggests that DataPrep.EDA generally improved participants' efficiency in performing EDA tasks. Factoring in dataset, we find that Pandas-profiling performs better in small datasets(M:3, SD:4.13) compared to a more complex one (M:1.1, SD:1.20, t(14)=−3.26062, $\rho < .0028$). As dataset complexity grows, Pandas-profiling fails to scale up. No participants finished all tasks, and 42% finished at most one task using Pandas-profiling. On the other hand, 35% of DataPrep.EDA participants finished all five tasks for the *delayed* dataset. We did not observe a dataset difference for DataPrep.EDA (t(14)=−0.66, $\rho < .51$), which suggests that it scaled well and might have pushed participants towards an efficiency ceiling.

In terms of the number of correct answers versus ground truth, participants who used DataPrep.EDA (M:3.72, SD:0.06) were ***2.2*** times more accurate compared to those using Pandas-profiling (M:1.70, SD:3.56, t(29)=2.791, $\rho < .001$), which suggests that DataPrep.EDA better assisted users in analyzing and reduced the risk of false discoveries. We again found that there was no significant difference detected between datasets (t(14)=.4156, $\rho < .1299$), however, as in completed tasks, we find that Pandas-profiling did a significantly better job for small datasets and failed to guide users for larger ones (t(14)=−1.27, $\rho < .00042$). These results are encouraging: DataPrep.EDA can help participants with different skill-levels complete many tasks with fewer errors.

As the number of completed tasks affects the amount of (in)correct answers, we used our relative accuracy metric. The average relative accuracy of DataPrep.EDA (M: .82 , SD: 0.07) among participants was ***1.5*** times higher than Pandas-profiling (M: .53 , SD: 1.35).

Considering expertise and dataset complexity (Figure 5.7), we find that users from both skill levels achieve similar relative accuracy in both datasets, but skilled participants did significantly better than novice participants only for Pandas-profiling in complex datasets. This suggests that DataPrep.EDA performed better at leveling skill differences and dataset complexity.

Qualitative feedback. In our post-survey we also asked participants to share comments and feedback to add context to the performance differences we observed. In our quantitative results, participants often referenced the responsiveness and efficiency of DataPrep.EDA ("fast and responsive") and took issue with the speed of Pandas-profiling ("it didn't work, took forever to process", "I would also like to use Pandas-profiling if the efficiency is not the bottleneck"). We also asked how more granular information affected their performance. Participants reflected that they felt more control ("I can find the necessary information very quickly, and that really helps a lot for me to solve the problems and questions very quickly", "felt like I had more control, simpler results") and accessible ("I find all the answers I need, and DataPrep.EDA is more easy to understand").

5.6 Conclusion & Future Work

In this work, we proposed a task-centric EDA tool design pattern and built such a system in Python, called DataPrep.EDA. We carefully designed the API in DataPrep.EDA, mapping common EDA tasks in statistical modeling to corresponding stats/plots. Additionally, DataPrep.EDA provides tools for automatically generating insights and providing guides. Through our implementation, we discussed several issues in building data processing pipelines using Dask and presented our solutions. We conducted a performance evaluation on 15 real-world data science datasets from Kaggle and a user study with 32 participants. The results showed that DataPrep.EDA significantly outperformed Pandas-profiling in terms of both speed and user experience.

We believe that task-centric EDA is a promising research direction. There are many interesting research problems to explore in the future. Firstly, there are some other EDA tasks for statistical modeling. For example, time-series analysis is a common EDA task in finance (e.g., stock price analysis). It would be interesting to study how to design a task-centric API for these tasks as well. Secondly, we notice that the speedup of DataPrep.EDA over Pandas tends to get small when IO becomes the bottleneck. We plan to investigate how to reduce IO cost using data compression techniques and column store. Thirdly, we plan to leverage sampling and sketches to speed up computation. The challenges are i) how to detect the scenarios of applying sampling/sketches; ii) how to notify users of the possible risk of sampling/sketches in a user-friendly way.

Chapter 6

Conclusion & Future Direction

In this chapter, we summarize the presented approaches and discuss some possible future directions.

6.1 Thesis Summary

In this thesis, we aim to accelerate human-in-the-loop data analytics. We observed that the challenge exists on both machine and human sides. From the machine side, there exists a gap between the limited hard resources and the massive volume of data. From the human side, the gap exists between the limited human attention and the enormous details that need to be handled to finish a task.

To accelerate machine processing, we developed two AQP systems with the same idea of combining multiple data summaries to achieve a better trade-off between answer quality and response time.

- *AQP++.* We first present AQP++, a unified framework to combine samples with pre-computed aggregations (stored in a cube). We identified two questions in building AQP++ (i.e., *aggregate identification* and *aggregate precomputation*) and propose both optimal solutions (under some assumptions) and heuristic approaches (for the general case). The experiment results show that AQP++ can improve the answer quality of sample-based estimation by 10X, without introducing too much cost.

- *SamComb.* AQP++ leverages cubes to improve the estimation quality. However, cube only supports range queries and has a high preprocessing cost. Therefore, we also study how to combine different types of samplers. We then propose SamComb, a bandit-based sampler combination framework. Given a budget (i.e., sample size), a query and a set of samplers, SamComb iteratively allocates the budget and decides how much budget should be allocated to each sampler. We propose two strategies adapted from MAB to balance the exploration and exploitation trade-off in budget allocation. We prove both two strategies can allocate most of the budget to the best sampler. After the budget allocation, SamComb then combines all estimators from samplers by allocating different weights. We studied two weight allocation strategies and proved that SamComb is asymptotically optimal under the proportional-to-size weight allocation.

To accelerate human analytics, we focused on the exploratory data analysis scenario and developed an EDA system with the idea of task-centric API design.

- *DataPrep.EDA.* We propose DataPrep.EDA to simplify the steps of conducting EDA in Python. We carefully design a set of declarative APIs to map common EDA tasks in statistical modelling to corresponding visualizations. Since each API represents a task, users can finish an EDA task with a single function call. In this way, users can pay attention to deciding the task to perform and leave the implementation details to the system. Our experiment results show that DataPrep.EDA significantly outperforms Pandas-profiling (a popular data-profiling tool in Python) in terms of both speed and user experience.

6.2 Future Direction

Increasing Supported Queries in AQP systems. While AQP systems have been studied for decades, they have not been widely used in practice. One possible reason may relate to their limited supported query types.

1. *Scope of Supported Queries.* It lacks a clear boundary between queries that an AQP system can support and queries that the system cannot support. It seems there is no systematic study on this problem. While the answer may be hard to know, we may choose another direction: support as many types of queries as possible. In the past, various synopses [73] have been proposed to support different types of queries. To answer as many queries as possible, the combination idea of this thesis can be applied: different synopsis can be combined to support more query types.

2. *Supporting Complex Queries.* It is very challenging for an AQP system to support complex queries such as join and nested queries. Recently, there seems to be a trend of applying advanced machine learning models in AQP. The idea is to model the data distribution (data-driven) or query distribution (query-driven), and predicts the query result. ML-based approaches have achieved some promising results in scenarios such as cardinality estimation, and it shows a potential to support complex queries [174, 106]. Although it seems promising, many challenging problems are still unsolved [105]. E.g., how to handle the data drift problem? How to guarantee the quality of the returned result?

ML-powered Analytics. Machine learning (ML) techniques have been successfully applied in real applications such as healthcare [81], image analysis [148] and natural language processing [128]. It also brings a trend in data analytics. Gartner uses the term “augmented analytics" to represent the approach *“that automates insights using machine learning and natural-language generation"*, and shows that it *“marks the next wave of disruption in the data and analytics market"* [145].

One question is that if we think ML techniques can be as smart as humans, can we leverage them to assist data analytics? In the following, we list some scenarios where ML may be applied to help accelerate human analytics.

1. *Insight Generation.* DataPrep.EDA is designed for the scenario where users know what task to perform. It is also possible that users do not have a specific task in mind and they just want to find some interesting facts about the dataset. To accelerate analytics in this scenario, modern BI tools can automatically recommend interesting insights from the dataset (e.g., QuickInsight in PowerBI [79]). Although the system can help humans explore interesting facts, the large search space makes it hard to achieve an interactive response. One idea to accelerate insight generation is to leverage approximate computation. If an insight does not appear in the result set, we may also draw the same conclusion using the approximate score of the insight.

2. *Explanation Generation.* When users explore the data, it is very common for them to find something unusual and then seek an explanation. E.g., why the income increase compared to last month? To accelerate this process, existing tools can automatically generate explanations of some interesting facts. For example, PowerBI has a feature to explain fluctuations in visualization. Many efforts have also been made in academia [114, 171]. Similar to the issue of insight generation, it is also challenging to generate explanations for large-scale data due to the large search space. One idea to solve this problem is to improve search efficiency. E.g., BOExplain [114] leverages bayesian optimization to explain inference queries.

3. *Natural Language Interaction.* The advanced natural language processing techniques make it possible to interact with the system in a natural language interface. Although some BI tools allow users to ask a natural language question and return a visualization as an answer (e.g., the ’Q & A’ feature of PowerBI), the currently supported questions are very simple, and the support of natural language is still at an early stage. Actually, natural language has the potential to become a unified interface for conducting data analytics. For example, users first ask “what is interesting about the data?". Then the system returns an insight that the average salary in a city is much higher than in other cities. After that, the user asks “why is the average salary in city X high?". Finally, the system receives the question and gives an explanation. In order to build such a system, there exist many challenges that need to be solved, such as handling the ambiguity of natural language queries, mapping the query to visual representation and managing the dialogue system [149].

Bibliography

[1] Big Data Benchmark. `http://amplab.cs.berkeley.edu/benchmark`.

[2] Loan dataset. `https://www.kaggle.com/skihikingkevin/online-p2p-lending`.

[3] TLC Trip Record Data. `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`.

[4] ACLED Asian Conflicts, 2015-2017, 2020.

[5] Adult Census Income, 2020.

[6] Alteryx: Automation that lets data speak and people think, 2020.

[7] Automobile Dataset, 2020.

[8] AutoViz: Automatically Visualize any dataset, any size with a single line of code, 2020.

[9] Basketball Players Stats per Season - 49 Leagues, 2020.

[10] Bitcoin Dataset, 2020.

[11] Chess Game Dataset (Lichess), 2020.

[12] Data science courses on edX, 2020.

[13] DataExplorer: Automate Data Exploration and Treatment, 2020.

[14] Default of Credit Card Clients Dataset, 2020.

[15] Diamonds, 2020.

[16] From Data to Viz, 2020.

[17] Heart Disease UCI, 2020.

[18] Hotel booking demand, 2020.

[19] IBM Data Science Professional Certificate, 2020.

[20] IBM SPSS Statistics: Easy-to-Use Data Analysis, 2020.

[21] JMP: Statistical Discovery From SAS, 2020.

[22] Kaggle: Your Machine Learning and Data Science Community, 2020.

[23] Lux: A Python API for Intelligent Visual Discovery, 2020.

[24] Microsoft Excel: Work together on Excel spreadsheets, 2020.

[25] Microsoft Power BI: Data Visualization, 2020.

[26] Pima Indians Diabetes Database, 2020.

[27] Python for Data Science and Machine Learning Bootcamp, 2020.

[28] Qlik: Data Analytics and Data Integration Solutions, 2020.

[29] Rain in Australia, 2020.

[30] SAS: Analytics, Artificial Intelligence and Data Management, 2020.

[31] Solar Radiation Prediction, 2020.

[32] splunk: The Data-to-Everything Platform, 2020.

[33] Suicide Rates Overview 1985 to 2016, 2020.

[34] Sweetviz: an open source Python library that generates beautiful, high-density visualizations to kickstart EDA (Exploratory Data Analysis) with a single line of code, 2020.

[35] Tableau: an interactive data visualization software company, 2020.

[36] The TIOBE Programming Community Index, 2020.

[37] The UC Berkeley Foundations of Data Science Course, 2020.

[38] TIBCO Spotfire Data Visualization and Analytics Software, 2020.

[39] Titanic: Machine Learning from Disaster, 2020.

[40] Top Women Chess Players, 2020.

[41] Koalas: pandas API on Apache Spark, 2021.

[42] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *The VLDB Journal*, 24(4):557–581, 2015.

[43] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *SIGMOD*, 2000.

[44] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD*, 1999.

[45] C. Adams, L. Alonso, B. Atkin, J. Banning, S. Bhola, R. Buskens, M. Chen, X. Chen, Y. Chung, Q. Jia, N. Sakharov, G. Talbot, N. Taylor, and A. Tart. Monarch: Google's planet-scale in-memory time series database. *Proc. VLDB Endow.*, 13(12):3181–3194, 2020.

[46] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. I. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *SIGMOD*, 2014.

[47] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[48] S. Agrawa. Multi-armed bandits and reinforcement learning course, lecture 3.

[49] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[50] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.

[51] D. Barbará and M. Sullivan. Quasi-cubes: Exploiting approximations in multidimensional databases. *SIGMOD Record*, 1997.

[52] L. Battle and J. Heer. Characterizing exploratory visual analysis: A literature review and evaluation of analytic provenance in tableau. In *Computer Graphics Forum*, volume 38, pages 145–159. Wiley Online Library, 2019.

[53] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[54] A. Bilogur. Missingno: a missing data visualization suite. *Journal of Open Source Software*, 3(22):547, 2018.

[55] Bokeh Development Team. *Bokeh: Python library for interactive visualization*, 2018.

[56] S. Brugman. Pandas-profiling: Exploratory Data Analysis for Python. `https://github.com/pandas-profiling/pandas-profiling`, 2019.

[57] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[58] Y. Cao and W. Fan. Data driven approximation with bounded resources. *PVLDB*, 10(9):973–984, 2017.

[59] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[60] S. K. Card and J. Mackinlay. The structure of the information visualization design space. In *Proceedings of VIZ'97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium*, pages 92–99. IEEE, 1997.

[61] C. Y. Chan and Y. E. Ioannidis. Hierarchical prefix cubes for range-sum queries. In *VLDB*, 1999.

[62] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. R. Narasayya. Overcoming limitations of sampling for aggregation queries. In *ICDE*, 2001.

[63] S. Chaudhuri, G. Das, and V. R. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *SIGMOD*, 2001.

[64] S. Chaudhuri, G. Das, and V. R. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2):9, 2007.

[65] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 1997.

[66] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In *SIGMOD*, 2017.

[67] S. Chaudhuri and V. Narasayya. TPC-D data generation with skew. `ftp.research.microsoft.com/users/viveknar/tpcdskew`.

[68] Y. Chen and Z. Ghahramani. Scalable discrete sampling as a multi-armed bandit problem. In *International Conference on Machine Learning*, pages 2492–2501. PMLR, 2016.

[69] Y. Chen and K. Yi. Two-level sampling for join size estimation. In *SIGMOD*, 2017.

[70] R. Chirkova and J. Yang. Materialized views. *Foundations and Trends in Databases*, 4(4):295–405, 2012.

[71] S. Chun, C. Chung, J. Lee, and S. Lee. Dynamic update cube for range-sum queries. In *VLDB*, 2001.

[72] S. Cohen, W. Nutt, and Y. Sagiv. Rewriting queries with arbitrary aggregation functions using views. *ACM Trans. Database Syst.*, 31(2):672–715, 2006.

[73] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[74] Z. Cui, S. K. Badam, M. A. Yalçin, and N. Elmqvist. Datasite: Proactive visual data exploration with computation of insight-based recommendations. *Information Visualization*, 18(2):251–267, 2019.

[75] Ç. Demiralp, P. J. Haas, S. Parthasarathy, and T. Pedapati. Foresight: Recommending visual insights. *arXiv preprint arXiv:1707.03877*, 2017.

[76] D. Deutch, A. Gilad, T. Milo, and A. Somech. Explained: explanations for eda notebooks. *Proceedings of the VLDB Endowment*, 13(12):2917–2920, 2020.

[77] V. Dibia and Ç. Demiralp. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications*, 39(5):33–46, 2019.

[78] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample + Seek: approximating aggregates with distribution precision guarantee. In *SIGMOD*, pages 679–694, 2016.

[79] R. Ding, S. Han, Y. Xu, H. Zhang, and D. Zhang. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the 2019 International Conference on Management of Data*, pages 317–332, 2019.

[80] C. E. Dyreson. Information retrieval from an incomplete data cube. In *VLDB*, 1996.

[81] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[82] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 10(10):1142–1153, 2017.

[83] V. Ganti, M. Lee, and R. Ramakrishnan. ICICLES: self-tuning samples for approximate query answering. In *VLDB*, 2000.

[84] S. Geffner, D. Agrawal, A. El Abbadi, and T. R. Smith. Relative prefix sums: An efficient approach for querying dynamic OLAP data cubes. In *ICDE*, 1999.

[85] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*, 1998.

[86] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.

[87] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[88] M. H. Hansen and W. N. Hurwitz. On the theory of sampling from finite populations. *The Annals of Mathematical Statistics*, 14(4):333–362, 1943.

[89] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[90] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, 1997.

[91] C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *SIGMOD*, 1997.

[92] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo. Vizml: A machine learning approach to visualization recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[93] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[94] C. Jermaine. Robust estimation with sampling and approximate pre-aggregation. In *VLDB*, pages 886–897, 2003.

[95] C. Jermaine and R. J. Miller. Approximate query answering in high-dimensional data cubes. In *SIGMOD*, 2000.

[96] C. M. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the DBO engine. In *SIGMOD*, 2007.

[97] R. Jin, L. Glimcher, C. Jermaine, and G. Agrawal. New sampling-based estimators for OLAP queries. In *ICDE*, 2006.

[98] S. Joshi and C. Jermaine. Materialized sample views for database approximation. In *ICDE*, 2006.

[99] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In *ICDE*, pages 472–483, 2014.

[100] N. Kamat and A. Nandi. A session-based approach to fast-but-approximate interactive data cube exploration. *ACM Trans. Knowl. Discov. Data*, 12(1):9:1–9:26, Feb. 2018.

[101] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 547–554, 2012.

[102] S. Kandula, K. Lee, S. Chaudhuri, and M. Friedman. Experiences with approximating queries in microsoft's production big-data clusters. *Proc. VLDB Endow.*, 12(12):2131–2142, 2019.

[103] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*, 2016.

[104] N. R. Katsipoulakis, A. Labrinidis, and P. K. Chrysanthis. Spear: Expediting stream processing with accuracy guarantees. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1105–1116. IEEE, 2020.

[105] K. Kim, J. Jung, I. Seo, W. Han, K. Choi, and J. Chong. Learned cardinality estimation: An in-depth study. In Z. Ives, A. Bonifati, and A. E. Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 1214–1227. ACM, 2022.

[106] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org, 2019.

[107] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, and T. Kraska. Stale view cleaning: Getting fresh answers from stale materialized views. *PVLDB*, 8(12):1370–1381, 2015.

[108] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *SIGMOD*, 2016.

[109] K. Li and G. Li. Approximate query processing: what is new and where to go? *Data Science and Engineering*, 3(4):379–397, 2018.

[110] X. Li, J. Han, Z. Yin, J. Lee, and Y. Sun. Sampling cube: a framework for statistical olap over sampling data. In *SIGMOD*, 2008.

[111] W. Liang, H. Wang, and M. E. Orlowska. Range queries in dynamic OLAP data cubes. *Data Knowl. Eng.*, 34(1):21–38, 2000.

[112] Q. Lin, W. Ke, J.-G. Lou, H. Zhang, K. Sui, Y. Xu, Z. Zhou, B. Qiao, and D. Zhang. Bigin4: Instant, interactive insight identification for multi-dimensional big data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 547–555, 2018.

[113] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[114] B. Lockhart, J. Peng, W. Wu, J. Wang, and E. Wu. Explaining inference queries with bayesian optimization. *Proc. VLDB Endow.*, 14(11):2576–2585, 2021.

[115] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 101–112. IEEE, 2018.

[116] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE transactions on visualization and computer graphics*, 13(6):1137–1144, 2007.

[117] A. Mahajan and D. Teneketzis. Multi-armed bandit problems. In *Foundations and Applications of Sensor Management*, pages 121–151. Springer, 2008.

[118] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska. Bao: Making learned query optimization practical. In G. Li, Z. Li, S. Idreos, and D. Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1275–1288. ACM, 2021.

[119] G. Moerkotte. Small materialized aggregates: A light weight index structure for data warehousing. In *VLDB*, 1998.

[120] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *CHI*, 2017.

[121] D. Moritz, C. Wang, G. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2019.

[122] B. Mozafari and N. Niu. A handbook for building an approximate query engine. *IEEE Data Eng. Bull.*, 38(3):3–29, 2015.

[123] B. Mozafari, J. Ramnarayan, S. Menon, Y. Mahajan, S. Chakraborty, H. Bhanawat, and K. Bachhav. Snappydata: A unified cluster for streaming, transactions and interactice analytics. In *CIDR*, 2017.

[124] I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *SIGMOD*, 1997.

[125] S. Nirkhiwale, A. Dobra, and C. M. Jermaine. A sampling algebra for aggregate estimation. *PVLDB*, 6(14):1798–1809, 2013.

[126] F. Olken. *Random Sampling from Databases.* PhD thesis, University of California at Berkeley, 1993.

[127] F. Olken and D. Rotem. Simple random sampling from relational databases. In *VLDB*, 1986.

[128] D. W. Otter, J. R. Medina, and J. K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Trans. Neural Networks Learn. Syst.*, 32(2):604–624, 2021.

[129] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4(11):1135–1145, 2011.

[130] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with metanome. *Proceedings of the VLDB Endowment*, 8(12):1860–1863, 2015.

[131] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: Universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1461–1476, 2018.

[132] Y. Park, A. S. Tajik, M. J. Cafarella, and B. Mozafari. Database learning: Toward a database that becomes smarter every time. In *SIGMOD*, pages 587–602, 2017.

[133] J. Peng, B. Ding, J. Wang, K. Zeng, and J. Zhou. One size does not fit all: A bandit-based sampler combination framework with theoretical guarantees. In Z. Ives, A. Bonifati, and A. E. Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 531–544. ACM, 2022.

[134] J. Peng, W. Wu, B. Lockhart, S. Bian, J. N. Yan, L. Xu, Z. Chi, J. M. Rzeszotarski, and J. Wang. Dataprep.eda: Task-centric exploratory data analysis for statistical modeling in python. In G. Li, Z. Li, S. Idreos, and D. Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 2271–2280. ACM, 2021.

[135] J. Peng, D. Zhang, J. Wang, and J. Pei. AQP++: connecting approximate query processing with aggregate precomputation for interactive analytics. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1477–1492, 2018.

[136] R. Peng. *Exploratory data analysis with R.* Lulu. com, 2012.

[137] D. Petersohn, W. W. Ma, D. J. L. Lee, S. Macke, D. Xin, X. Mo, J. E. Gonzalez, J. M. Hellerstein, A. D. Joseph, and A. G. Parameswaran. Towards scalable dataframe systems. *CoRR*, abs/2001.00888, 2020.

[138] A. Pol and C. Jermaine. Relational confidence bounds are easy with the bootstrap. In *SIGMOD*, pages 587–598, 2005.

[139] N. Potti and J. M. Patel. DAQ: A new paradigm for approximate query processing. *PVLDB*, 8(9):898–909, 2015.

[140] C. Proteau. Guide to performance and tuning: Query performance and sampled selectivity, 2004.

[141] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[142] K. Rong. *Improving computational and human efficiency in large-scale data analytics*. PhD thesis, Stanford University, USA, 2021.

[143] F. Rusu, C. Qin, and M. Torres. Scalable analytics model calibration with online aggregation. *IEEE Data Eng. Bull.*, 38(3):30–43, 2015.

[144] F. Rusu, F. Xu, L. L. Perez, M. Wu, R. Jampani, C. Jermaine, and A. Dobra. The DBO database system. In *SIGMOD*, pages 1223–1226, 2008.

[145] R. Sallam, C. Howson, and C. J. Idoine. Augmented analytics is the future of data and analytics. *Gartner, Inc*, 27, 2017.

[146] H. J. Seltman. Experimental design and analysis, 2012.

[147] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner. Presto: SQL on everything. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pages 1802–1813. IEEE, 2019.

[148] D. Shen, G. Wu, and H.-I. Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221, 2017.

[149] L. Shen, E. Shen, Y. Luo, X. Yang, X. Hu, X. Zhang, Z. Tai, and J. Wang. Towards natural language interfaces for data visualization: A survey. *arXiv preprint arXiv:2109.03506*, 2021.

[150] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *VLDB*, 1998.

[151] A. Shukla, P. Deshpande, J. F. Naughton, and K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In *VLDB*, 1996.

[152] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. zenvisage: Effortless visual data exploration. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, 2016.

[153] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. SciBORQ: Scientific data management with bounds on runtime and quality. In *CIDR*, 2011.

[154] S. Singh. *Advanced Sampling Theory With Applications: How Michael"" Selected"" Amy*, volume 2. Springer Science & Business Media, 2003.

[155] C. J. Skinner. Probability proportional to size (pps) sampling. *Wiley StatsRef: Statistics Reference Online*, pages 1–5, 2014.

[156] A. Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[157] M. Staniak and P. Biecek. The landscape of r packages for automated exploratory data analysis. *arXiv preprint arXiv:1904.02101*, 2019.

[158] B. Tang, S. Han, M. Yiu, R. Ding, and D. Zhang. Extracting top-k insights from multi-dimensional data. pages 1509–1524, 05 2017.

[159] N. Tierney. visdat: Visualising whole data frames. *Journal of Open Source Software*, 2(16):355, 2017.

[160] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, volume 8, page 2182. NIH Public Access, 2015.

[161] E. Veach and L. J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In S. G. Mair and R. Cook, editors, *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1995, Los Angeles, CA, USA, August 6-11, 1995*, pages 419–428. ACM, 1995.

[162] D. Vengerov, A. C. Menck, M. Zaït, and S. Chakkappen. Join size estimation subject to filter conditions. *Proc. VLDB Endow.*, 8(12):1530–1541, 2015.

[163] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, 1999.

[164] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, pages 469–480, 2014.

[165] M. Waskom and the seaborn development team. mwaskom/seaborn, Sept. 2020.

[166] H. Wickham and G. Grolemund. *R for data science: import, tidy, transform, visualize, and model data.* " O'Reilly Media, Inc.", 2016.

[167] K. Wongsuphasawat, Y. Liu, and J. Heer. Goals, process, and challenges of exploratory data analysis: An interview study. *CoRR*, abs/1911.00568, 2019.

[168] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics*, 22(1):649–658, 2015.

[169] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2648–2659, 2017.

[170] T. Wright. Exact optimal sample allocation: More efficient than neyman. *Statistics & Probability Letters*, 129:50–57, 2017.

[171] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.

[172] S. Wu, B. C. Ooi, and K. Tan. Continuous sampling for online aggregation over multiple queries. In *SIGMOD*, 2010.

[173] J. N. Yan, Z. Gu, and J. M. Rzeszotarski. Tessera: Discretizing data analysis workflows on a task level. In *CHI '21: CHI Conference on Human Factors in Computing Systems, Yokohama, Japan, May 8–13, 2021*, pages 1–15. ACM, 2021.

[174] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep unsupervised cardinality estimation. *Proc. VLDB Endow.*, 13(3):279–292, 2019.

[175] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In E. M. Nahum and D. Xu, editors, *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010.* USENIX Association, 2010.

[176] K. Zeng, S. Agarwal, and I. Stoica. iOLAP: managing uncertainty for efficient incremental OLAP. In *SIGMOD*, 2016.

[177] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, pages 277–288, 2014.

Appendix A

Supplementary Material for AQP++

A.1 Proofs

Proof of Lemma 1

Consider a user query q:

SELECT $f(A)$ FROM $\mathcal{D}$ WHERE Condition_1,

and a precomputed aggregate query *pre*:

SELECT $f(A)$ FROM $\mathcal{D}$ WHERE Condition_2.

If AQP supports the aggregation function f, query q and *pre* can be estimated using AQP, i.e., $\hat{q}(\mathcal{S})$ and $\hat{pre}(\mathcal{S})$. Since $pre(\mathcal{D})$ is a constant, AQP++ can use Equation 3.4 to get the estimatation of q.

Proof of Lemma 2

Consider a user query q:

SELECT $f(A)$ FROM $\mathcal{D}$ WHERE Condition_1,

and a precomputed aggregate query *pre*:

SELECT $f(A)$ FROM $\mathcal{D}$ WHERE Condition_2.

If AQP can estimate their answers unbiasedly, then we have $q(\mathcal{D}) = \mathbf{E}[\hat{q}(\mathcal{S})]$ and $pre(\mathcal{D}) = \mathbf{E}[\hat{pre}(\mathcal{S})]$. Based on Equation 3.4, AQP++'s estimator returns $pre(\mathcal{D}) + (\hat{q}(\mathcal{S}) - \hat{pre}(\mathcal{S}))$. We can prove that its expect value is equal to the true value:

$$\begin{aligned}
&\mathbf{E}\Big[pre(\mathcal{D}) + \big(\hat{q}(\mathcal{S}) - \hat{pre}(\mathcal{S})\big)\Big] \\
&= \mathbf{E}[pre(D)] + \big(\mathbf{E}[\hat{q}(\mathcal{S})] - \mathbf{E}[\hat{pre}(\mathcal{S})]\big) \\
&= pre(\mathcal{D}) + \big(q(\mathcal{D}) - pre(\mathcal{D})\big) \\
&= q(\mathcal{D})
\end{aligned}$$

Proof of Lemma 4

Consider a sequence of i.i.d values $\mathcal{D} = \{a_1, a_2, \cdots, a_n\}$. We define $\mathcal{D}_{pre} = \{G_{pre}(a_i) \mid i \in [1, n]\}$, where $G_{pre}(a_i)$ take a_i as input and returns a_i if a_i satisfies the pre's condition; otherwise, 0. We define $\mathcal{D}_q = \{G_q(a_i) \mid i \in [1, n]\}$, where $G_q(a_i)$ take a_i as input and returns a_i if a_i satisfies the q's condition; otherwise, 0. We define $\mathcal{D}_d = \mathcal{D}_{pre} - D_{pre} = \{x_i - y_i | x_i \in \mathcal{D}_{pre}, y_i \in \mathcal{D}_q, i \in [1, n].\}$ Based on the confidence interval for a SUM query in Example 4, we can deduce that $error(q, pre) = \lambda N \sqrt{\frac{\texttt{VAR}(\mathcal{D}_d)}{n}}$. Since $\texttt{VAR}(\mathcal{D}_d) = E[\mathcal{D}_d^2] - E[\mathcal{D}_d]^2$, let us compute $E[\mathcal{D}_d^2]$ and $E[\mathcal{D}_d]^2$ separately.

Firstly, we will compute $E[\mathcal{D}_d]^2$. We have $E[\mathcal{D}_d] = E[\mathcal{D}_{pre} - D_q] = E[\mathcal{D}_{pre}] - E[\mathcal{D}_q]$. Since each value in $\mathcal{D}$ is i.i.d, the sum of the values within any range is proportional to the length of the range. Let α be the percentage of the values in $\mathcal{D}$ satisfying pre's condition but not q's, β be the percentage of the values in $\mathcal{D}$ satisfying q's condition but not pre's, and γ be the percentage of the values in $\mathcal{D}$ satisfying both pre's and q's condition. Then, we have $E[\mathcal{D}_{pre}] = (\alpha + \gamma)E[\mathcal{D}]$ and $E[\mathcal{D}_q] = (\beta + \gamma)E[\mathcal{D}]$. Hence,

$$E[\mathcal{D}_d]^2 = (\alpha - \beta)^2 E[\mathcal{D}]^2 \tag{A.1}$$

Secondly, we will compute $E[\mathcal{D}_d^2]$. We have $\mathcal{D}_{preq} = \{G_{preq}(a_i) \mid i \in [1, n]\}$, where $G_{preq}(a_i)$ take a_i as input and returns a_i if a_i satisfies the pre's condition as well as the q's condition; otherwise, 0. Similar to the idea of computing $E[\mathcal{D}_d]^2$, we obtain $E[\mathcal{D}_{pre}^2] = (\alpha + \gamma)E[\mathcal{D}^2]$, $E[\mathcal{D}_q^2] = (\beta + \gamma)E[\mathcal{D}^2]$ and $E[\mathcal{D}_{preq}^2] = \gamma E[\mathcal{D}^2]$. Hence, we can get:

$$E[\mathcal{D}_d^2] = (\alpha + \beta)E[\mathcal{D}^2] \tag{A.2}$$

Combining Equation A.1 and Equation A.2, we can derive $error(q, pre) = \lambda N \sqrt{\frac{\sigma^2}{n}}$, where $\sigma^2 = (\alpha + \beta)E[\mathcal{D}^2] - (\alpha - \beta)^2 (E[\mathcal{D}])^2$.

Given q, our goal is to find pre with the minimal error, which is equivalent to find the minimal σ^2. Denote the selectivity of q as θ. The following two equations always hold:

$$\theta \geq \beta \tag{A.3}$$

$$0 \leq \alpha + \theta \leq 1 \tag{A.4}$$

Let us consider pre in P^+ in the following three cases:

(1) Case 1: pre satisfies $\alpha \geq \beta + \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2}$. we will prove that $\sigma^2 - error(q, \phi) \geq 0$, which means $\phi \in P^-$ is the optimal pre query. Actually, we have $\sigma^2 \geq (\alpha - \beta)E[\mathcal{D}^2] - (\alpha - \beta)^2 E[\mathcal{D}]^2$. Hence, $\sigma^2 - error(q, \phi) \geq (\alpha - \beta - \theta)E[\mathcal{D}^2] - (\alpha - \beta - \theta)(\alpha - \beta + \theta)E[\mathcal{D}]^2$.

Since $\alpha - \beta \geq \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2} \geq \frac{1}{2}$, and $\theta \leq 1 - \alpha \leq \frac{1}{2}$ (based on Equation A.4), we have $\alpha - \beta - \theta \geq 0$. Since $\alpha - \beta + \theta \leq 1 - \beta \leq 1$, we can get $E[\mathcal{D}^2] - (\alpha - \beta + \theta)E[\mathcal{D}]^2 \geq E[\mathcal{D}^2] - E[\mathcal{D}]^2 \geq 0$. Combine it with $\alpha - \beta - \theta \geq 0$, we can derive $\sigma^2 - error(q, \phi) \geq 0$.

(2) Case 2: pre satisfies $\beta \geq \alpha + \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2}$. We will prove that $\phi \in P^-$ is the optimal pre, i.e., $\sigma^2 - error(q, \phi) \geq 0$. Since $\beta \geq \alpha$, we can get $(\alpha - \beta)^2 \leq \beta^2$. Besides, we also have $\alpha + \beta \geq \beta$. Then, we can derive $\sigma^2 = (\alpha + \beta)E[\mathcal{D}^2] - (\alpha - \beta)^2 E[\mathcal{D}]^2 \geq \beta E[\mathcal{D}^2] - \beta^2 E[\mathcal{D}]^2$.

Hence, $\sigma^2 - error(q, \phi) \geq (\beta - \theta)E[\mathcal{D}^2] - (\beta - \theta)(\beta + \theta)E[\mathcal{D}]^2$.

Since we have $\beta \geq \alpha + \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2} \geq \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2}$ and $\theta \geq \beta$ (Equation A.3), we can get $\beta + \theta \geq 2\beta \geq \frac{E[\mathcal{D}^2]}{E[\mathcal{D}]^2}$. Hence, we have $E[\mathcal{D}^2] - (\beta + \theta)E[\mathcal{D}]^2 \leq E[\mathcal{D}^2] - E[\mathcal{D}]^2 \frac{E[\mathcal{D}^2]}{E[\mathcal{D}]^2} = 0$. Combine it with $\beta - \theta \leq 0$, we could derive $\sigma^2 - error(q, \phi) \geq 0$.

(3) Case 3: pre satisfies $\beta < \alpha + \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2}$ and $\alpha < \beta + \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2}$. We will prove that a query $pre \in P^-$ will have the smallest error. If regard σ^2 as a quadratic function of α, then the turn point is $\beta + \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2}$. Since $\alpha < \beta + \frac{E[\mathcal{D}^2]}{2E[\mathcal{D}]^2}$, we can get that for a fixed β, the error is monotonically increasing w.r.t. α. Similarly, for a fixed α, the error is monotonically increasing w.r.t. β. Now given a query q, there are five possible positions that pre query can be. Let x and y denote the lowest point and the highest point of q query. Let pre_x and pre_y denote the lowest point and the highest point of pre query, respectively. Let pre_{opt} denote the optimal pre with smallest error.

(a) position 1: $pre_l \geq x$ and $pre_h \leq y$. In this case, $alpha = 0$ and $\beta = pre_x - x + y - pre_y$. Then the smallest β will get when $pre_x = h_x$ and $pre_y = l_y$. Hence, $pre_{opt} \in P^-$.

(b) position 2: $pre_x < x$ and $pre_y \leq y$. In this case, $\alpha = x - pre_x$ and $\beta = y - pre_y$. Then the smallest α and β is got when $pre_x = l_x$ and $pre_y = l_y$. Hence, $pre_{opt} \in P^-$.

(c) position 3: $pre_x \geq x$ and $pre_y > y$. In this case, $\alpha = pre_y - y$ and $\beta = pre_x - x$. Then the smallest α and β is got when $pre_x = h_x$ and $pre_y = h_y$. Hence, $pre_{opt} \in P^-$.

(d) position 4: $pre_x < x$ and $pre_y > y$. In this case, $\beta = 0$ and $\alpha = x - pre_x + pre_y - y$. Then the smallest α is got when $pre_x = l_x$ and $pre_y = h_y$. Hence, $pre_{opt} \in P^-$.

(e) position 5: $pre = \phi$. Since $\phi \in P^-$, we also have $pre_{opt} \in P^-$.

Proof of Lemma 4

According to Lemma 4, we only need to prove

$$\max_{q \in Q} \min_{pre \in P^-_{eq}} error(q, pre) = \lambda N \sqrt{\frac{\sigma^2_{eq}}{n}}.$$

Let θ denote q's selectivity.

(1) If $\theta > \frac{1}{k}$. In this case, there exists two points h_x and l_y inside query q. We will prove that when x and y are the middle points of $l_x h_x$ and $l_y h_y$, we can get $\max_{q \in Q} \min_{pre \in P_{eq}^-} error(q, pre) = \lambda N \sqrt{\frac{\sigma_{eq}^2}{n}}$, where $\sigma_{eq}^2 = \frac{1}{k} E[\mathcal{D}^2] - \frac{1}{k^2}(E[\mathcal{D}])^2$.

First we will prove that when x and y are the middle points, we have $\sigma_{opt}^2 = \sigma_{eq}^2$. Actually, there are five pre we can choose: $pre = AGG(l_x : h_y)$, $pre = AGG(h_x : l_y)$, $pre = AGG(l_x : l_y)$, $pre = AGG(h_x : h_y)$, and $pre = \phi$.

(a) If $pre = AGG(l_x : h_y)$, then $\alpha = \frac{1}{k}$ and $\beta = 0$, thus $\sigma^2 = \sigma_{eq}^2$.

(b) If $pre = AGG(h_x : l_y)$, similar to (a) we can get $\sigma^2 = \sigma_{eq}^2$.

(c) If $pre = AGG(l_x : l_y)$, we have $\alpha = \frac{1}{2k}$ and $\beta = \frac{1}{2k}$, hence $\sigma^2 = \frac{1}{k} E[\mathcal{D}^2] \geq \sigma_{eq}^2$.

(d) If $pre = AGG(h_x : h_y)$, similar to (c) we can get $\sigma^2 = \frac{1}{k} E[\mathcal{D}^2] \geq \sigma_{eq}^2$.

(e) If $pre = \phi$, we have $\sigma^2 = \theta E[\mathcal{D}^2] - \theta^2 E[\mathcal{D}]^2$. Since $\frac{1}{k} \leq \theta \leq 1 - \frac{1}{k}$, we can get $\sigma^2 \geq \sigma_{eq}^2$.

Hence, $\sigma_{opt}^2 = \sigma_{eq}^2$.

Now we will prove that for any query q, we have $\sigma_{opt}^2 \leq \sigma_{eq}^2$. Suppose $l = |h_x l_y|$ and $L = |l_x h_y|$.

(a) When $\theta - l \leq \frac{1}{k}$, for $pre = h_x l_y$, we have $\alpha = 0$ and $\beta = \theta - l \leq \frac{1}{k}$. Then, $\sigma^2 = \beta E[\mathcal{D}^2] - \beta^2 E[\mathcal{D}]^2 \leq \sigma_{eq}^2$. Hence, we can get $\sigma_{opt}^2 \leq \sigma^2 \leq \sigma_{eq}^2$.

(b) When $\theta - l > \frac{1}{k}$, for $pre = l_x h_y$, we have $\alpha = L - \theta = l + \frac{2}{k} - \theta < \frac{1}{k}$ and $\beta = 0$. Then $\sigma^2 = \alpha E[\mathcal{D}^2] - \alpha^2 E[\mathcal{D}]^2 < \sigma_{eq}^2$. Hence, we can get $\sigma_{opt}^2 \leq \sigma^2 < \sigma_{eq}^2$.

(2) If $\theta \leq \frac{1}{k}$, we will prove that the query-template error cannot be larger than $\lambda N \sqrt{\frac{\sigma_{eq}^2}{n}}$. When using ϕ to answer the query, the query's variance is $\sigma^2 = \theta E[\mathcal{D}^2] - \theta^2 (E[\mathcal{D}])^2$. Since $\theta \leq \frac{1}{k}$ and σ^2 is monotonically increasing w.r.t θ, we have $\sigma^2 \leq \sigma_{eq}^2$. Thus, $error(q, \phi) = \lambda N \sqrt{\frac{\sigma^2}{n}} \leq \lambda N \sqrt{\frac{\sigma_{eq}^2}{n}}$. Since $\phi \in P_{eq}^-$, we obtain $\min_{p \in P_{eq}^-} error(q, p) \leq error(q, \phi) \leq \lambda N \sqrt{\frac{\sigma_{eq}^2}{n}}$.

Proof of Lemma 5

In order to prove the lemma, we only need to construct a single bad query $q' \in Q$ such that $error(q', P) \geq \lambda N \sqrt{\frac{\sigma_{eq}^2}{n}}$. Since the precomputed queries are not evenly chosen, there must exist two intervals such that the sum of their lengths is larger than $\frac{2N}{k}$. Construct a query $q' = \text{SUM}(x : y)$, where x and y are the middle points of the two intervals, respectively. Suppose $|l_x h_x| = 2a$ and $|l_y h_y| = 2b$. Then we have $\frac{1}{k} \leq a + b \leq 1 - \theta$ and $\theta \geq a + b \geq \frac{1}{k}$. There are five possible pre queries for q':

(a) $pre = AGG(l_x : h_y)$: we have $\beta = 0$ and $\alpha = a + b$. Since $\frac{1}{k} \leq a + b \leq 1 - \theta \leq 1 - \frac{1}{k}$, we have $\sigma^2 = (a+b) E[\mathcal{D}^2] - (a+b)^2 E[\mathcal{D}]^2 \geq \frac{1}{k} E[\mathcal{D}^2] - \frac{1}{k^2} E[\mathcal{D}]^2 = \sigma_{eq}^2$.

(b) $pre = AGG(h_x : l_y)$: similar to (a), we have $\sigma^2 = (a+b) E[\mathcal{D}^2] - (a+b)^2 E[\mathcal{D}]^2 \geq \sigma_{eq}^2$.

(c) $pre = AGG(l_x : l_y)$: we have $\alpha = a$ and $\beta = b$. Then $\sigma^2 = (a+b) E[\mathcal{D}^2] - (a-b)^2 E[\mathcal{D}]^2 \geq (a+b) E[\mathcal{D}^2] - (a+b)^2 E[\mathcal{D}]^2 > \sigma_{eq}^2$.

(d) $pre = AGG(h_x : h_y)$: similar to (c), we can get $\sigma^2 \geq \sigma_{eq}^2$.

(e) $pre = \phi$: we have $\alpha = 0$ and $\beta = \theta$. Since $\frac{1}{k} \leq \theta \leq 1 - \frac{1}{k}$, we have $\sigma^2 = \theta E[\mathcal{D}^2] - \theta^2 E[\mathcal{D}]^2 \geq \sigma_{eq}^2$.

Now, for all five pre queries, we have $\sigma^2 \geq \sigma_{eq}^2$. Hence the query error of q' is $\min_{pre \in P^-} error(q', pre) \geq \lambda N \sqrt{\frac{\sigma_{eq}^2}{n}}$.

Proof of Theorem 1

Based on Lemmas 4 and 5, we can easily deduce that the query-template error of Q w.r.t. P_{eq} is minimum. That is,

$$P_{eq} = \arg\min_{P} \max_{\substack{q = \text{SUM}(x:y) \\ 1 \leq x < y \leq N}} error(q, P).$$

Hence, P_{eq} is an optimal BP-Cube.

Proof of Lemma 6

Due to the space limit, we just give the proof sketch. $P^- \setminus \{\phi\} \subseteq P^+$, then $error(q, P) = \min_{p \in P^+} error(q, p) \leq \min_{p \in P^- \setminus \{\phi\}} error(q, p)$. We can see that $P^- \setminus \{\phi\}$ consists of four precomputed queries that lead to four ways to estimate the sum. As shown in Section 3.6.1, we can choose the pre query that leads to $\min\{\frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(A_{L_x})}, \frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(A_{\bar{L}_x})}\}$ and $\min\{\frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(A_{L_y})}, \frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(A_{\bar{L}_y})}\}$. Then $error(q, P)$ would be $\frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(X + Y)}$ or $\frac{\lambda N}{\sqrt{n}} \cdot \sqrt{\text{Var}(X - Y)}$, where $X = A_{L_x}$ if $\text{Var}(A_{L_x}) \leq \text{Var}(A_{\bar{L}_x})$, otherwise $X = A_{\bar{L}_x}$. Y has a similar meaning with X. Then, based on Cauchy-Schwarz inequality $\sqrt{\text{Var}(X \pm Y)} \leq \sqrt{\text{Var}(X)} + \sqrt{\text{Var}(Y)}$, we can derive the lemma.

A.2 Preprocessing Cost Analysis

We analyze the time complexity of our aggregate-precomputation technique in this part.

In the first stage, we need to determine which BP-Cube needs to be precomputed. This stage is only executed on a sample $\mathcal{S}$. The total time complexity of this stage is dominated by determining the BP-Cube's shape because it needs to run hill climbing algorithms for multiple times (denote the number of the times by m) in order to plot an error profile for each dimension. The hill climbing algorithm is an iterative algorithm. Let $iter$ denote the number of iterations. Each iteration takes a linear time of $\mathcal{O}(n)$. Thus, the time complexity of plotting a single error profile is $\mathcal{O}(m \cdot iter \cdot n)$. Since we need to construct d error profiles, the total time complexity is $\mathcal{O}(d \cdot m \cdot iter \cdot n)$. Note that here n is the sample size, which is orders of magnitude smaller than the data size. In the experiments, we set $m = 20$ by default, and found that $iter$ is on average smaller than 20.

In the second stage, we need to precompute the $\mathsf{BP\text{-}Cube}$ obtained from the first stage. Ho et al. [91] proposed an efficient algorithm to do so. The algorithm needs to scan the full data once to initialize a d-dimensional array of the size of $\prod_{i=1}^{d} k_i$. The time complexity of this step is $\mathcal{O}(N \cdot \log k)$ and the I/O cost is $\mathcal{O}(\mathcal{D})$. Next, the algorithm scans the array for d times and the final d-dimensional array is the $\mathsf{BP\text{-}Cube}$ that we want to precompute. The time complexity of this step is $\mathcal{O}(d \cdot k)$. Since $\mathsf{BP\text{-}Cube}$ is often small, we assume that it can be put in memory, thus this step does not involve any I/O cost. To sum up, the total time complexity is $\mathcal{O}(N \cdot \log k + d \cdot k)$ and the total I/O cost is $\mathcal{O}(\mathcal{D})$. Since the entire $\mathsf{P\text{-}Cube}$ consists of $\prod_{i=1}^{d} |\mathsf{dom}(C_i)|$ cells and a $\mathsf{BP\text{-}Cube}$ only contains $k (\ll \prod_{i=1}^{d} |\mathsf{dom}(C_i)|)$ cells, $\mathsf{AQP{+}{+}}$ incurs much less preprocessing cost than AggPre in terms of both space usage and running time.

Appendix B

Supplementary Material for SamComb

B.1 Proof of Lemma 7

Proof. Since $\mathbb{S}_{i^*}$ is the best sampler, based on the definition of sampler quality, we have

$$\mathsf{var}(\mathcal{D}_i) \geq \mathsf{var}(\mathcal{D}_{i^*})$$

for all $i \in [1, k]$.

By incorporating it into Equation (4.8), we derive the following:

$$\frac{1}{\sum_{i=1}^{k} \frac{n_i}{\mathsf{var}(\mathcal{D}_i)}} \geq \frac{1}{\sum_{i=1}^{k} \frac{n_i}{\mathsf{var}(\mathcal{D}_{i^*})}} = \frac{1}{\frac{n}{\mathsf{var}(\mathcal{D}_{i^*})}}.$$

In the above inequality, the left-hand side denotes the variance of the combined estimator of *any* budget allocation strategy and the right-hand side represents the variance of the combined estimator when allocating all the budget n to the best sampler $\mathbb{S}_{i^*}$. We can see the right-hand side is always smaller than or equal to the left-hand side, thus the lemma is proved. $\square$

B.2 Proof of Lemma 8

We first prove that when $t \geq cK/d^2$, the probability of choosing a sub-optimal sampler $\mathbb{S}_i$ is at most $\mathcal{O}(\frac{1}{t})$.

The proof is a simple adaption of [49] and we use similar notations. Let $T_j(n)$ be the number of batches of sampler j at timestamp n, and let $V_{j,T_j(n)}$ be the $\hat{V}ar(\mathcal{D}_j)$ with $T_j(n)$ chunks. Let

$$x_0 = \frac{1}{2K} \sum_{t=1}^{n} \epsilon_t \tag{B.1}$$

The probability of choosing sampler j at timestamp n is:

$$Pr(I_n = j) \leq \frac{\epsilon_n}{K} + (1 - \frac{\epsilon_n}{K})Pr(\hat{V}_{j,T_j(n-1)} \leq \hat{V}^*_{T^*(n-1)})$$

we have:

$$\begin{aligned}
&Pr\{\hat{V}_{j,T_j(n-1)} \leq \hat{V}^*_{T^*(n-1)}\} \\
=&Pr\{\hat{V}_{j,T_j(n-1)} - (V_j - \frac{\Delta_j}{2}) \leq \hat{V}^*_{T^*(n-1)} - (V^* + \frac{\Delta_j}{2})\} \\
\leq&Pr\{\hat{V}_{j,T_j(n-1)} \leq V_j - \frac{\Delta_j}{2} \text{ or } \hat{V}_{j,T_j(n-1)} \geq V^* + \frac{\Delta_j}{2}\} \\
=&Pr\{\hat{V}_{j,T_j(n-1)} \leq V_j - \frac{\Delta_j}{2}\} + Pr\{\hat{V}_{j,T_j(n-1)} \geq V^* + \frac{\Delta_j}{2}\}
\end{aligned}$$

The analysis of the left term and the right term are the same. We have:

$$\begin{aligned}
&Pr\{\hat{V}_{j,T_j(n)} \leq V_j - \frac{\Delta_j}{2}\} \\
=&\sum_{t=1}^{n} Pr(T_j(n) = t \text{ and } \hat{V}_{j,t} \leq V_j - \frac{\Delta_j}{2}) \\
=&\sum_{t=1}^{n} Pr(T_j(n) = t | \hat{V}_{j,t} \leq V_j - \frac{\Delta_j}{2}) \cdot Pr(\hat{V}_{j,t} \leq V_j - \frac{\Delta_j}{2}) \\
&\text{(by Hoeffding bound)} \\
\leq&\sum_{t=1}^{n} Pr(T_j(n) = t | \hat{V}_{j,t} \leq V_j - \frac{\Delta_j}{2}) \cdot e^{\frac{-t\Delta_j^2}{2d_j^2}} \\
\leq&\sum_{t=1}^{x_0} Pr(T_j(n) = t | \hat{V}_{j,t} \leq V_j - \frac{\Delta_j}{2}) \cdot e^{\frac{-t\Delta_j^2}{2d_j^2}} \\
&+ \sum_{t=x_0+1}^{n} Pr(T_j(n) = t | \hat{V}_{j,t} \leq V_j - \frac{\Delta_j}{2}) \cdot e^{\frac{-t\Delta_j^2}{2d_j^2}} \\
\leq&\sum_{t=1}^{x_0} Pr(T_j(n) = t | \hat{V}_{j,t} \leq V_j - \frac{\Delta_j}{2}) + \frac{2d_j^2}{\Delta_j^2} e^{\frac{-\Delta_j^2 x_0}{2d_j^2}} \\
&\text{(since } \sum_{t=x_0+1}^{\infty} e^{-ct} \leq \frac{1}{c} e^{-cx_0}) \\
\leq&\sum_{t=1}^{x_0} Pr(T_j^R(n) \leq t | \hat{V}_{j,t} \leq V_j - \frac{\Delta_j}{2}) + \frac{2d_j^2}{\Delta_j^2} e^{\frac{-\Delta_j^2 x_0}{2d_j^2}} \\
\leq&x_0 \cdot Pr(T_j^R(n) \leq x_0) + \frac{2d_j^2}{\Delta_j^2} e^{\frac{-\Delta_j^2 x_0}{2d_j^2}}
\end{aligned} \tag{B.2}$$

where the last line is because each sampler is chosen at random and independent of previous choice. Since

$$E[T_j^R(n)] = \frac{1}{K}\sum_{t=1}^{n}\epsilon_t \tag{B.3}$$

and

$$Var[T_j^R(n)] = \sum_{i=1}^{n}\frac{\epsilon_t}{K}(1-\frac{\epsilon_t}{K}) \leq \frac{1}{K}\sum_{t=1}^{n}\epsilon_t \tag{B.4}$$

, by Bernstein's equality we get:

$$Pr\{T_j^R(n) \leq x_0\} \leq e^{-x_0/5} \tag{B.5}$$

Finally we lower bound x_0. For $n \geq n' = cK/d^2$, $\epsilon_n = cK/(d^2 n)$ and we have:

$$\begin{aligned} x_0 &= \frac{1}{2K}\sum_{t=1}^{n}\epsilon_t \\ &= \frac{1}{2K}\sum_{t=1}^{n'}\epsilon_t + \frac{1}{2K}\sum_{t=n'+1}^{n}\epsilon_t \\ &\geq \frac{n'}{2K} + \frac{c}{d^2}ln\frac{n}{n'} \\ &\geq \frac{c}{d^2}ln\frac{nd^2e^{1/2}}{cK} \end{aligned} \tag{B.6}$$

Combing the above equations and note that $d \leq \frac{\Delta_j}{d_j}$ and $\frac{1}{x^2}e^{-x^2}$ is monotonically decreasing we finally get:

$$\begin{aligned} &Pr(I_n = j) \leq \frac{\epsilon_n}{K} + 2x_0e^{-x_0/5} + \frac{4d_j^2}{\Delta_j^2}e^{-\frac{\Delta_j^2 x_0}{2d_j^2}} \\ &\leq \frac{c}{d^2 n} + 2(\frac{c}{d^2}ln\frac{(n-1)d^2e^{1/2}}{cK})(\frac{cK}{(n-1)d^2e^{1/2}})^{c/(5d^2)} \\ &+ \frac{4e}{d^2}(\frac{cK}{(n-1)d^2e^{1/2}})^{c/2} \end{aligned} \tag{B.7}$$

As stated in [49], when $c > 5$, the above bound is actually $\mathcal{O}(\frac{1}{t})$.

After we derive the bound of the probability of selecting a sub-optimal sampler, next we prove the expected batches allocated to it. We have:

$$\begin{aligned} b_{i,(n-1)/b} &= 1 + \frac{1}{K}\cdot\frac{cK}{d^2} + \sum_{t=cK/d^2}^{(n-1)/b} Pr(I_t = i) \\ &\leq 1 + \frac{1}{K}\cdot\frac{cK}{d^2} + \sum_{t=cK/d^2}^{(n-1)/b}\mathcal{O}(\frac{1}{t}) \end{aligned} \tag{B.8}$$

Note that $\sum_{t=cK/d^2}^{(n-1)/b} \mathcal{O}(\frac{1}{t}) \leq \mathcal{O}(\ln n)$. Hence, we finally proved that the budget allocated to a sub-optimal sampler is $b \cdot b_{i,(n-1)/b} \leq \mathcal{O}(\ln n)$.

B.3 Proof of Lemma 10

The proof is similar to [48]. It contains two parts.

First Part. In the first part, we prove that: at any time t, if the number of batches allocated to a sub-optimal sampler $\mathbb{S}_i$ satisfying: $\mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2}$, then $lcb_{i,t} > lcb_{i^*,t}$ with a probability at least $1 - 2\sigma_t$, i.e.:

$$Pr(I_{t+1} = i | \mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2}) \leq 2\sigma_t, \tag{B.9}$$

where I_{t+1} is the selected sampler at timestamp $t+1$.

When $\mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2}$, i.e., $u_i \sqrt{\frac{\ln \frac{2}{\sigma_t}}{2\mathbb{S}_i.\mathsf{batchNum}_t}} \leq \frac{\Delta_i}{2}$, we have:

$$\begin{aligned} lcb_{i,t} &= \hat{V}ar(\mathcal{D}_i)_t - u_i \sqrt{\frac{\ln \frac{2}{\sigma_t}}{2\mathbb{S}_i.\mathsf{batchNum}_t}} \\ &\geq \hat{V}ar(\mathcal{D}_i)_t - \frac{\Delta_i}{2} \end{aligned} \tag{B.10}$$

Besides, with prob. at least $1 - \sigma_t$, we have:

$$\begin{aligned} \hat{V}ar(\mathcal{D}_i)_t &\geq Var(\mathcal{D}_i) - u_i \sqrt{\frac{\ln \frac{2}{\sigma_t}}{2\mathbb{S}_i.\mathsf{batchNum}_t}} \\ &\geq Var(\mathcal{D}_i) - \frac{\Delta_i}{2} \end{aligned} \tag{B.11}$$

Combing Equation B.10 and Equation B.11, we could get:

$$\begin{aligned} lcb_{i,t} &\geq Var(\mathcal{D}_i) - \Delta_i \\ &= Var(\mathcal{D}_{i^*}) \end{aligned} \tag{B.12}$$

with prob. at least $1 - \sigma_t$.

Since $Var(\mathcal{D}_{i^*}) \geq lcb_{i^*,t}$ with prob. at least $1 - \sigma_t$, we could finally get $lcb_{i,t} \geq lcb_{i^*,t}$ with prob. at least $(1 - \sigma_t)^2 \geq 1 - 2\sigma_t$.

Second Part. Then, in the second part, we prove that:

$$\mathbb{E}[\mathbb{S}_i.\mathsf{batchNum}_t] \leq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2} + 2\sum_{j=1}^{t} \sigma_j \tag{B.13}$$

We first prove that:

$$\mathbb{E}[\sum_{t=0}^{T-1} \mathbb{1}(I_{t+1} = i, \mathbb{S}_i.\mathsf{batchNum}_t < \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2})] \leq \frac{2u_i^2 \ln \frac{2}{\sigma_{T-1}}}{\Delta_i^2} \tag{B.14}$$

Let us consider the contradiction case: we assume the indicator equals to 1 more than $\frac{2u_i^2 \ln \frac{2}{\sigma_{T-1}}}{\Delta_i^2}$ times. Then there must exist a time stamp $\tau \leq T-1$, such that:

$$\mathbb{E}[\sum_{t=0}^{\tau} \mathbb{1}(I_{t+1} = i, \mathbb{S}_i.\mathsf{batchNum}_t < \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2})] = \frac{2u_i^2 \ln \frac{2}{\sigma_{T-1}}}{\Delta_i^2} \tag{B.15}$$

Hence, for any $t \geq \tau + 1$, $\mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_{T-1}}}{\Delta_i^2}$. It implies $\mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2}$ as $\sigma_{T-1} \leq \sigma_t$. Based on the proof of the first part, the indicator will not be 1 for any time stamp $t \geq \tau + 1$ with high probability, which contradicts our assumption.

Now, we can prove that:

$$\begin{aligned}
&\mathbb{E}[\mathbb{S}_i.\mathsf{batchNum}_n] \\
=&\mathbb{E}[\sum_{t=0}^{n-1} \mathbb{1}(I_{t+1} = i)] \\
=&\mathbb{E}[\sum_{t=0}^{n-1} \mathbb{1}(I_{t+1} = i, \mathbb{S}_i.\mathsf{batchNum}_t < \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2})] \\
&+ \mathbb{E}[\sum_{t=0}^{n-1} \mathbb{1}(I_{t+1} = i, \mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2}))] \\
\leq&\frac{2u_i^2 \ln \frac{2}{\sigma_{n-1}}}{\Delta_i^2} + \mathbb{E}[\sum_{t=0}^{n-1} \mathbb{1}(I_{t+1} = i, \mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2}))] \\
=&\frac{2u_i^2 \ln \frac{2}{\sigma_{n-1}}}{\Delta_i^2} + \sum_{t=0}^{n-1} Pr(I_{t+1} = i, \mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2})) \\
=&\frac{2u_i^2 \ln \frac{2}{\sigma_{n-1}}}{\Delta_i^2} + \sum_{t=0}^{n-1} Pr(I_{t+1} = i | \mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2})) \\
&\cdot Pr(\mathbb{S}_i.\mathsf{batchNum}_t \geq \frac{2u_i^2 \ln \frac{2}{\sigma_t}}{\Delta_i^2})) \\
\leq&\frac{2u_i^2 \ln \frac{2}{\sigma_{n-1}}}{\Delta_i^2} + 2\sum_{t=0}^{n-1} \sigma_t
\end{aligned} \tag{B.16}$$

To make $\sum \sigma_j$ converged, we set σ_j as a constant (e.g., 0.5) when $j < 2$ and $\sigma_j = 2/j^2$ when $j > 2$. Then, the above bound is in the order of $\frac{4u_i^2 \ln n-1}{\Delta_i^2} + constant$, which is $\mathcal{O}(\ln n)$.

B.4 Proof of Lemma 11

Under optimal weight allocation, we have:

$$\mathsf{var}(q(\psi)) = \frac{1}{\sum_{i=1}^{k} \frac{n_i}{\mathsf{var}(\mathcal{D}_i)}} \tag{B.17}$$

We first analyze the bound of $\frac{\mathsf{var}(q(\psi))}{\mathsf{var}(q(\psi_*))}$. Let $f(n)$ denotes the upper bound of the budget allocated to all sub-optimal samplers, i.e., $\sum_{i\neq i^*} n_i \leq f(n)$, we have:

$$\begin{aligned}
\frac{\mathsf{var}(q(\psi))}{\mathsf{var}(q(\psi_*))} =& \frac{\frac{1}{\frac{n_{i^*}}{\mathsf{var}(\mathcal{D}_{i^*})}+\sum_{i\neq i^*}\frac{n_i}{\mathsf{var}(\mathcal{D}_i)}}}{\mathsf{var}(\mathcal{D}_{i^*})/n} \\
=& \frac{n}{n_{i^*} + \mathsf{var}(\mathcal{D}_{i^*}) \cdot \sum_{i\neq i^*} \frac{n_i}{\mathsf{var}(\mathcal{D}_i)}} \\
\leq& \frac{n}{n - \sum_{i\neq i^*} n_i + \frac{\mathsf{var}(\mathcal{D}_{i^*})}{\max_i\{\mathsf{var}(\mathcal{D}_i)\}} \cdot \sum_{i\neq i^*} n_i} \\
=& \frac{n}{n - (1 - \frac{\mathsf{var}(\mathcal{D}_{i^*})}{\max_i\{\mathsf{var}(\mathcal{D}_i)\}}) \cdot \sum_{i\neq i^*} n_i} \\
\leq& \frac{n}{n - (1 - \frac{\mathsf{var}(\mathcal{D}_{i^*})}{\max_i\{\mathsf{var}(\mathcal{D}_i)\}}) \cdot f(n)}
\end{aligned} \tag{B.18}$$

We then analyze the relative error of ϵ-greedy and LCB. For ϵ-greedy and LCB, $f(n)$ is $\mathcal{O}(\ln n)$. Hence, we finally have:

$$\begin{aligned}
gap(q(\psi)) =& \frac{\mathsf{var}(q(\psi))}{\mathsf{var}(q(\psi_*))} - 1 \\
\leq& \frac{\mathcal{O}(\ln n)}{n - \mathcal{O}(\ln n)}
\end{aligned}, \tag{B.19}$$

which shows asymptotic optimality.

B.5 Proof of Lemma 12

Under proportional-to-size allocation, we have:

$$\mathsf{var}(q(\psi)) = \frac{1}{n^2} \sum_{i=1}^{k} n_i \mathsf{var}(\mathcal{D}_i) \tag{B.20}$$

Let $f(n)$ denotes the upper bound of the budget allocated to all sub-optimal samplers. Then we have:

$$
\begin{aligned}
\frac{\mathsf{var}(q(\psi))}{\mathsf{var}(q(\psi_*))} &= \frac{\frac{1}{n^2}\sum_{i=1}^{k} n_i \mathsf{var}(\mathcal{D}_i)}{\mathsf{var}(\mathcal{D}_{i^*})/n} \\
&= \frac{(n - \sum_{i \neq i^*} n_i) \cdot \mathsf{var}(\mathcal{D}_{i^*}) + \sum_{i \neq i^*} n_i \cdot \mathsf{var}(\mathcal{D}_i)}{n \cdot \mathsf{var}(\mathcal{D}_{i^*})} \\
&\leq \frac{(n - \sum_{i \neq i^*} n_i) \cdot \mathsf{var}(\mathcal{D}_{i^*}) + \max_j \{\mathsf{var}(\mathcal{D}_j)\} \cdot \sum_{i \neq i^*} n_i}{n \cdot \mathsf{var}(\mathcal{D}_{i^*})} \\
&= \frac{n\mathsf{var}(\mathcal{D}_{i^*}) + (\max_i \mathsf{var}(\mathcal{D}_i) - \mathsf{var}(\mathcal{D}_{i^*})) \sum_{i \neq i^*} n_i}{n \cdot \mathsf{var}(\mathcal{D}_{i^*})} \\
&\leq 1 + \frac{(\max_i \mathsf{var}(\mathcal{D}_i) - \mathsf{var}(\mathcal{D}_{i^*})) f(n)}{n \cdot \mathsf{var}(\mathcal{D}_{i^*})}
\end{aligned}
$$

Since $f(n) \leq \mathcal{O}(\ln n)$, we finally have:

$$
\begin{aligned}
gap(q(\psi)) &= \frac{\mathsf{var}(q(\psi))}{\mathsf{var}(q(\psi_*))} - 1 \\
&\leq \frac{\mathcal{O}(\ln n)}{n},
\end{aligned}
\tag{B.21}
$$

which shows asymptotic optimality.