IMPROVING COMPUTATIONAL AND HUMAN EFFICIENCY IN LARGE-SCALE DATA ANALYTICS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Kexin Rong
August 2021

This dissertation is online at: https://purl.stanford.edu/nc796rp3408

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Philip Levis, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Peter Bailis

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Matei Zaharia

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Network telemetry, sensor readings, and other machine-generated data are growing exponentially in volume. Meanwhile, the computational resources available for processing this data – as well as analysts' ability to manually inspect it – remain limited. As the gap continues to widen, keeping up with the data volumes is challenging for analytic systems and analysts alike. This dissertation introduces systems and algorithms that focus the limited computational resources and analysts' time in modern data analytics on a subset of relevant data. The dissertation comprises two parts that focus on improving the computational and human efficiency in data analytics, respectively.

In the first part of this dissertation, we improve the computational efficiency of analytics by combining precomputation and sampling techniques to select a subset of data that contributes the most to query results. We demonstrate this concept with two approximate query processing systems. PS^3 approximates aggregate SQL queries with weighted, partition-level samples based on precomputed summary statistics, whereas HBE approximates kernel density estimations using precomputed hash indexes as smart data samplers. Our evaluation shows that both systems outperform uniform sampling, the best-known result for these queries, with practical precomputation overheads. PS^3 enables a 3 to 70× speedup under the same accuracy as uniform partition sampling, with less than 100 KB of storage overhead per partition; HBE offers up to a 10× improvements in query time compared to the second-best method with comparable precomputation time.

In the second part of this dissertation, we improve the human efficiency of analytics by automatically identifying and summarizing unusual behaviors in large data streams to reduce the burden of manual inspections. We demonstrate this approach through two monitoring applications for machine-generated data. First, ASAP is a visualization operator that automatically smooths time series in monitoring dashboards to highlight large-scale trends and deviations. Compared to presenting the raw time series, ASAP decreases users' response time for identifying anomalies by up to 44.3% in our user study. We subsequently describe FASTer, an end-to-end earthquake detection system that we built in collaboration with seismologists at Stanford University. By pushing down domain-specific filtering and aggregation into the analytics workflows, FASTer significantly improves the speed and quality of earthquake candidate generation, scaling the analysis from three months of data from a single sensor to ten years of data over a network of sensors.

The contributions of this dissertation have had real-world impact. ASAP has been incorporated into open-source tools such as Graphite, TimescaleDB Toolkit, and NPM module downsample. ASAP has also directly inspired an auto smoother for the real-time dashboards at the monitoring service Datadog. FASTer is open-source and has been used by researchers worldwide. Its improved scalability has enabled the discovery of hundreds of new earthquake events near the Diablo Canyon nuclear power plant in California.

Acknowledgments

When I started graduate school, I had limited research experience, so I did not know what to expect and was unsure whether I would make it through. However, I was fortunate to meet a group of wonderful mentors, collaborators, and friends who have helped make graduate school one of the most memorable journeys in my life.

I would like to thank my advisor, Peter Bailis. Peter played a critical role in introducing me to research. His enthusiasm for problems that are both intellectually interesting and grounded in real-world challenges has also profoundly shaped my research. During our first project, he encouraged me to sign up to speak at an industrial conference, which later became key to obtaining adoption. When there were no existing channels to contact potential users, Peter showed us how to reach out proactively and build new ones from scratch. Peter has also always believed in me and encouraged me to step outside my comfort zone even when I doubted myself. I owe much of my academic and personal growth during my PhD to him.

I would also like to thank my co-advisor, Philip Levis, for his constant guidance and support throughout my PhD. With his rich experiences and logical thinking, Phil always pointed me towards clarity when I became lost in project details or overall directions. Phil is also a meticulous researcher who constantly challenged me to think deeply about the assumptions, design experiments thoughtfully, and present ideas precisely. He spent time teaching me how to create legible tables and graphs early in my PhD, and I adhere to the same guidelines today. I also had the pleasure of working with Phil to help grow Stanford's undergraduate research program CURIS, along with Professor Michael Bernstein and PhD student Griffin Dietz. This was a rewarding experience for me and contributed positively to my decision to be an educator.

I would like to thank the members of my oral committee, Moses Charikar, Matei Zaharia, and Greg Beroza. Moses is the friendly theorist who often presents himself at Info lunches and is there to help. He always makes time for me whenever I reach out to chat about a random idea that I had, and makes me feel included even when the discussion delves deep into theory realm. Although I have not had the opportunity to collaborate with Matei, he has, many times, provided me valuable feedback on my work, connected me with potential industry collaborators, and selflessly helped with my job search. Getting to know Greg and his lab was probably the most unexpected and fun adventure

during my PhD. Thank you, Greg, for inviting us on the hunt for new earthquakes.

I would like to thank Hector Garcia-Molina, who has been the heart of the InfoLab and a kind mentor to many of us. Hector was our go-to person for critical feedback on research, and also for chocolates, InfoLab swags, and professional headshots. The fun trip reports and inspiring practice talks at Info lunches were some of my very first impressions of graduate school. I also want to thank my undergraduate mentors Adam Wierman and Steven Low. I did not think about applying to graduate schools until I was already halfway through my third year at Caltech. When I nervously reached out to them for help, they immediately made an effort to take me on their research projects. I would not be here today if it were not for them.

I would like to thank my wonderful collaborators who taught me many new things and helped me improve as a researcher and collaborator. I had the chance to work with the amazing undergraduate students Justin Chen and Hashem Elezabi, PhD students Clara Yoon, Karianne Bergen, Paris Syminelakis, Firas Abuzaid, Edward Gan, Sahaana Suri as well as my mentors at Microsoft Research Srikanth Kandula and Yao Lu. My collaborators are some of the nicest and most hardworking people that I have met: Clara patiently helped us label at least thousands of earthquake events, if not more; Paris and I had some heated discussions over the email that lasted through the Christmas break; after I lost my access to Microsoft data, Srikanth became my personal VPN, helped run my scripts on the clusters, and forwarded me the error messages back and forth.

I would like to thank my fellow lab mates in the FutureData group and the SING group: Firas Abuzaid, Hudson Ayers, Lingjiao Chen, Holly Chiang, Cody Coleman, Trevor Gale, Edward Gan, Luke Hsiao, Raejoon Jung, Daniel Kang, Fiodar Kazhamiaka, Omar Khattab, Kevin Kiningham, Peter Kraft, Deepak Narayanan, Deepti Raghavan, Keshav Santhanam, Geet Sethi, Chinmayee Shah, Sahaana Suri, Kaisheng Tai, Pratiksha Thaker, James Thomas, Todd Warszawski, Francis Yan and Gina Yuan. They have significantly increased my productivity, happiness, and fitness during my PhD. It turns out that the only thing that I got right regarding my expectations for graduate school was the fact that I would be surrounded by a group of talented peers. What I did not expect though is how much I enjoyed being around them for work and fun. Some of my fondest memories with the group include tagging along on the afternoon walks to the Coupa Cafe even when I do not need coffee, secretly passing around someone's birthday card to get it signed by the lab, turning the school's free personal training program into a FutureData workout group, playing volleyball at the DAWN retreat, and getting trapped by rain at the Iguazu airport. I would also like to acknowledge my office mates in Gates 433—Sahaana, Edward, and Daniel–with whom I spent many hours bouncing research ideas, critiquing paper drafts, getting food on the weekend, and supporting each other in highs and lows. I am grateful for having been in one of the most supportive lab environments I could have wished for throughout my PhD.

I would like to thank my parents, Jiangang Rong and Yunxiu Zhang, for their unconditional love and support. As first-generation college students, they truly appreciate the power of education.

They were the reason I first became interested in programming, and they have always worked hard to provide me with the best resources for education, even though it means that I am thousands of miles away from them. This is my tenth year studying in the United States, also the tenth year that I am missing the family reunions and important holidays back home. Thank you, mom and dad, for always being my biggest cheerleaders and for always having my back so that I can be fearless in the pursuit of my dreams. I am also fortunate to have shared this remarkable journey with my fiancé Danfei Xu, whom I met at the PhD program. Danfei has been primarily responsible for providing me with emotional support, honest feedback, lots of laughs, and amazing adventures. It is hard to believe how much each of us has grown as a researcher and person, and I look forward to the many more adventures we will embark on.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Data volumes are growing exponentially. Much of this growth is driven by an increasing number of automated data sources, such as sensors, devices, and large-scale internet services. For example, Google collects more than 2.5 terabytes of data per second from its infrastructure and applications as of 2019 [6]; an average of one petabyte of sensor data was recorded and processed per day at CERN in 2017 [122]. International Data Corporation projects that data volumes will experience a compound annual growth rate of 23% and that "the amount of data created over the next five years will be greater than twice the amount of data created since the advent of digital storage" [263].

Meanwhile, the computational resources available for processing this data remain limited. Due to constraints imposed by power delivery and dissipation, CPU performances have already plateaued, and even multi-core processors do not promise exponentially growing computation [272]. This gap between data volumes and the available computational resources is further exacerbated by the increased diversity and complexity of data analytics. In addition to SQL queries, users are increasingly interested in sophisticated analyses from the statistics and machine learning communities such as data modeling and anomaly detection, many of which have a complexity that grows quadratically with the size of the dataset or combinatorially with the problem dimensions.

Further down the analytics pipeline, analysts also have a hard time keeping pace with the data volumes. As Herbert Simon once said, "the wealth of information means a dearth of something else: a scarcity of whatever it is that information consumes" [296]. Each piece of information presented to the analysts consumes their time and attention. At present, analysts are already struggling to keep up their manual analyses and inspections with the growing data volumes. For example, through conversations with engineers working at web-scale companies, we learned that despite the large amount of data collected and stored in their monitoring systems, the engineers routinely access less than 6% of the collected data [29].

As the growth of data continues to outpace computational resources and analysts' attention, processing all generated data is increasingly untenable. Therefore, data analytics systems need to scale

sublinearly with the data size in their processing resources. In this dissertation, we develop systems and algorithms to make efficient use of limited computational resources and analysts' attention by focusing on a subset of relevant and important data via novel uses of summaries and synopses. The dissertation comprises two parts, which address processing bottlenecks in computational resources and analysts' attention, respectively. We begin with an overview of the two bottlenecks before presenting our contributions.

1.1 Dealing with Limited Computational Resources

The first part of this dissertation focuses on the gap between growing data volumes and limited computational resources. Two approaches to bridging this gap exist: improving the efficiency of the hardware to enable more data to be processed with fewer computation cycles or attempting to achieve the same results by processing only a subset of the data. Decades of research have led to significant developments using the first approach, but this progress has recently slowed. In the future, it is unlikely that improvements in hardware efficiency will be able to handle the growth of data volumes. Instead, this dissertation explores the second approach of processing less data. Specifically, we seek to focus limited computational resources on a small subset of the data that contributes significantly to query results.

The database community has explored many ideas to reduce the amount of data to be processed for large-scale analytics. For example, classic indexing data structures can quickly filter out data points that are irrelevant to the query [37, 115, 282]. Researchers have also studied how to avoid scanning all data points *relevant* to the query using Approximate Query Processing (AQP) techniques [4, 8, 70, 131, 151, 202]. AQP techniques aim to provide accurate approximations to true query answers at a fraction of the query costs, thus allowing users to trade off a small loss in result quality for potentially large improvements in query performance. Queries that might require hours or minutes using query engines that compute over all relevant records can be accomplished in less than a second when processed with AQP techniques. For latency-sensitive applications, such as interactive data exploration and visualization, this trade-off between performance and accuracy is acceptable and often desirable.

To avoid scanning all data points associated with the query, AQP systems often use either a sample of the dataset or precomputed summaries. Figure 1.1 illustrates the two typical AQP system architectures. The first approach, which we refer to as sampling-based architectures, is to collect a random sample of the dataset at query time and estimate query answers using the samples. In this method, the cost and quality of the query are only a function of the sample size, independent of the dataset size. Sampling-based architectures require no prior knowledge of the query workload. However, the sample size required to produce good approximations can differ greatly across workloads and datasets.

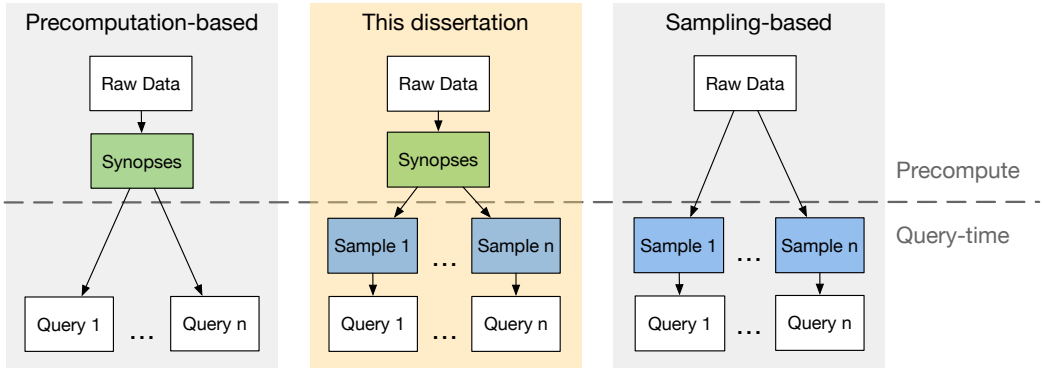


Figure 1.1: Illustrations of the traditional precomputation-based and sampling-based AQP system architectures, as well as the hybrid architecture studied in this dissertation.

The second approach, which we refer to as precomputation-based architectures, is to precompute compact summaries (i.e., synopses [82]) or query answers and process queries by combining the precomputed results. Since precomputed summaries and results are usually much smaller than the original dataset, this approach can offer significant performance improvements at query time. Unlike sampling, precomputation techniques are closely tied to query types and workloads. For example, quantile queries and distinct value queries require two different summaries; precomputed biased samples leverage knowledge of the workload, such as the sets of columns used.

The key design trade-off in AQP systems is between the precomputation cost and query performance. Existing AQP systems tend to fall into two extremes on this trade-off spectrum. On one extreme, taking random samples at query time requires no precomputation but is inaccurate, especially for skewed datasets. Since it may be necessary to take a large number of samples to achieve usable results, the performance benefits are limited. On the other extreme, exhaustively precomputing and storing all possible query answers provides fast and accurate answers. However, since different queries require different precomputations, the precomputations can incur large computation time and memory overheads. As a result, for many analytics tasks, users must choose between large performance improvements with large precomputation overheads or small performance improvements with random sampling.

1.2 Dealing with Limited Analyst Attention

The second part of this dissertation focuses on the gap between growing data volumes and limited analysts' attention. We investigate this gap in the context of monitoring systems.

The prevalence of automated data sources such as sensors, devices, and infrastructure has led to increasing demands for monitoring. Large-scale Internet services aim to remain highly available to users regardless of unexpected failures in the hardware and software infrastructure. Manufacturers

wish to gain real-time insights into their millions of Internet of Things devices in warehouses and assembly lines. Geologists measure ground movements of the earth to detect unusual seismic activities. All the above use cases require 1) the continuous collection and charting of the underlying systems' key performance metrics and 2) the ability to identify problems quickly as they arise. Failure to do so can lead to severe outcomes: outages of Amazon S3 services have impacted thousands of other services, and earthquakes cause thousands of deaths and injuries per year. In response to the large monitoring needs, several new industry systems for collecting, storing, and analyzing these sensor data have emerged in recent years.

The gap between data volumes and analysts' attention is of particular importance in monitoring applications. On the one hand, automated data sources can generate data at an overwhelming rate. For example, scientific experiments such as the Large Hadron Collider deploy millions of sensors to collect data, while Google ingests over 2.5 terabytes of data per second purely for monitoring purposes. On the other hand, analysts often need to manually inspect and verify the results generated by monitoring systems to take appropriate actions.

As a concrete example, at Microsoft's cloud infrastructure teams, engineers monitor the performance metrics from the millions of deployment configurations of their services and applications (e.g., application version × device type × network type × location) [2]. To detect unusual events such as high message sending failure rates in real time, engineers rely on a combination of computationally efficient static rules and thresholds. A simple rule can be to automatically classify the performance metrics in the top 1 percentile as outliers. However, with the large amounts of metrics collected, returning even a tiny fraction of the raw data can overwhelm end-users, who must manually inspect each result to determine whether it corresponds to an actual bug in the application.

We believe that the Microsoft example represents the experiences of many users of monitoring systems – it is much easier to collect data than to make sense of it, and the analyses are often bottlenecked by their own ability to sort through data points and configurations manually. As each result presented to the end-user consumes their time and attention, monitoring systems can not afford to simply return raw data records to users. Efficiently utilizing analysts' time requires careful considerations of the content as well as the format of the presentation to maximize the utility of each result shown.

While analysts can not manually inspect large amounts of data, machines can – monitoring systems can filter, highlight, and summarize data before it reaches a user. While different applications may require different solutions, as a general principle, we think that analytics systems should improve human efficiency in interpreting data by focusing the end-users' attention on a few important results. To achieve this goal, systems can leverage appropriate computation primitives such as filtering and aggregation to automatically explore the configuration and data space and summarize and rank the results on behalf of users.

1.3 Overview of Contributions

As the gap between the abundant data volumes and the scarce processing resources continues to widen, data analytics systems must evolve to address these new processing bottlenecks. Building upon prior work, this dissertation makes the following claim:

Thesis Statement: Data analytics systems that 1) combine precomputation and query-time sampling to improve computational efficiency and 2) automatically detect and summarize unusual behaviors to improve human efficiency scale to keep up with increasing data volumes from automated data sources such as sensors and devices.

The rest of this section discusses the contributions made in this dissertation to support this claim and provides a summary of the main results.

Part I: Improving Computational Efficiency

In the first part of this dissertation, we present AQP systems that improve computational efficiency by focusing limited computational resources on a subset of the data that contribute most to the query results. Unlike existing solutions, our designs achieve large performance improvements with practical precomputation overheads via a hybrid AQP system architecture that combines precomputed synopses with query-time sampling techniques (Figure 1.1, middle). The idea is to leverage the precomputation phase to gather information about the dataset that can then be used to select the data points that are top contributors to the answers at query time. Previous studies have also recognized the benefits of connecting sampling- and precomputation-based architectures [246]. Our contribution is to show that for some common analytics tasks, combining precomputation and sampling leads to the first results that outperform random sampling in terms of query performance with moderate precomputation overheads.

Specifically, we develop two AQP systems based on this hybrid architecture for aggregate queries. The goal of these systems is to approximate the results of queries that take the form of a summation of values from data points. The first is a system PS^3 (Partition Selection with Summary Statistics), which enables weighted partition-level sampling to approximate aggregate SQL queries in big-data clusters. The second system approximates kernel density estimations (KDEs) with random or biased samples for high-dimensional data. Both systems leverage precomputed indexes and summaries to help distinguish the relevance of the data points at query time. The following subsections describe the two systems in greater detail.

PS^3: Approximate Aggregate SQL Queries with Non-uniform Partition Samples. Large-scale production query processing systems, such as Spark [22], F1 [292], SCOPE [58], store data in large partitions (tens and hundreds of megabytes) and only support access at a coarse, partition-level

granularity. Approximating query answers with row-level samples in these systems is inefficient and can require reads of many partitions. In contrast, a *partition-level* sample requires I/O costs that are proportional to the sampling fraction. In fact, due to their appealing performance, partition-level samples are already supported in commercial databases such as Hive [47], Oracle [239], and Snowflake [299]. However, it is not known how to compute weighted samples over partitions, which is necessary because a uniform partition-level sample can lead to large errors in the estimates due to correlations between the rows in a partition.

Chapter 3 introduces PS^3, the first AQP system to enable weighted partition sampling to approximate aggregate SQL queries in big-data clusters. PS^3 targets read-only and append-only data stores, and explicitly chooses to improve query performance without requiring shuffling or reorganization of the dataset. PS^3 adopts the hybrid architecture and leverages precomputed partition-level summary statistics to improve the relevance of the sampled partitions at query time. This design raises two questions of what summaries statistics to compute and how to use them.

To determine what statistics to compute, we reference systems such as Spark [255] and Oracle Database [238], which already maintain partition-level summary statistics such as the maximum and minimum values of a column for query optimization purposes. Following similar design considerations for memory and storage footprints, we propose a set of sketches – measures, heavy hitters, distinct values, and histograms – to generate summary statistics during precomputation. These sketches are both cheap enough to compute and store and sufficiently discriminative to support decisions such as whether the partition contributes large values to the query answer. To answer the second question of how to use the precomputed statistics to improve partition sampling, we develop novel algorithms to assess the similarity (via clustering) and importance (via learning) of partitions based on statistics at query time.

Our experiments on several real-world datasets demonstrate that to achieve the same relative error as uniform partition sampling, PS^3 offers a 2.7× to 70× reduction in the number of partitions read, and the statistics stored per partition require fewer than 100 KB.

HBE: Approximate Kernel Density Estimation with Hashing. KDE provides normalized and differentiable probability densities that are useful in many applications, including visualization [195], domain science [153, 271, 315] and density-based outlier detection [100]. Despite its usefulness, the complexity of density computation grows quadratically with the size of the dataset. This work aims to, with some precomputation, efficiently estimate the KDE at a query point with $(1 \pm \epsilon)$ multiplicative accuracy.

Our work builds upon recent theoretical developments that have led to the first approximation algorithm, Hashing-Based-Estimator (HBE), with worst-case asymptotic improvements over random sampling in high dimensions [63]. Following the hybrid architecture design, HBE precomputes a specialized hash index to enable query-time sampling of data points that contribute heavily to the density estimates with higher probability. However, these advances are largely theoretical to

concerns such as large precomputation overheads and diminishing gains in non-worst-case datasets. In Chapter 4, we translate these theoretical ideas into a practical AQP system for KDE that demonstrates empirical improvements over competing methods.

To do so, we first reduce the superlinear precomputation time and memory of the hashing via sketching. The sketch allows us to perform precomputation on a small subset of the dataset while preserving the properties of the hash index. Second, while HBE is provably better than random sampling in worst-case scenarios, it can underperform random sampling on specific datasets. Accordingly, we introduce a cost-based optimizer that compares the sampling efficiency of the two methods on a per-dataset basis to select the more efficient query plan. Finally, we relax the conditions of the sampling procedure introduced in the original work, thus reducing the constant overhead by at least an order of magnitude.

Our experiments show that the system offers up to a 10× improvement in the average time required to evaluate the kernel density on a query point for a range of synthetic and real-world datasets. With the sketching procedure, the precomputation cost is reduced by at least 300× for the datasets we experimented with and therefore remains competitive against alternatives.

Part II: Improving Human Efficiency

In the second part of this dissertation, we present two monitoring applications that improve human efficiency by helping focus end-users' attention on important behaviors in the large data volumes. Both applications target time series data collected from sensors and devices. The first is a visualization operator ASAP (Automatic Smoothing for Attention Prioritization) designed to automatically smooth time series to highlight significant trends and deviations. The second is an earthquake detection system, FASTer, which identifies potential earthquake candidates by searching and aggregating similar patterns across sensors.

The focus on reducing analysts' workload has significantly improved the usability of both applications, as reflected by the real-world usages and adoptions: ASAP has been incorporated into several open-source tools and monitoring systems, and FASTer has enabled new scientific discoveries. The following subsections present the two applications in greater detail.

ASAP: Automatic Smoothing in Time Series Visualization. Application developers, site operators, and DevOps engineers rely on the charting and visualization of key performance metrics, usually presented in the form of time series, to perform monitoring, health checks, alerting, and analyses of unusual events such as failures [44, 160]. However, the efficient visualization of time series data remains a challenge. Since many metrics are inherently noisy, presenting the raw data can lead to unreadable graphs. As data volumes increase, even small-scale fluctuations in data values can obscure overall trends and behaviors.

Chapter 5 introduces a visualization operator ASAP designed to "distort" visualizations (e.g., by selecting an appropriate time scale for the presentation that hides local noise and fluctuations)

to prioritize users' attention towards significant deviations. ASAP, therefore, differs from existing smoothing techniques that are designed to produce visually indistinguishable representations of the original data (e.g., [173, 318]). To achieve ASAP's design goal, we propose the simple strategy of smoothing time series visualizations as much as possible while preserving large-scale deviations. This raises two key questions. First, how can we quantitatively assess the quality of a given visualization in removing small-scale variations and highlighting significant deviations? Second, how can we use the quantitative metrics to produce high-quality visualizations at scale?

To quantitatively assess the visualization quality, we propose a novel problem formulation combining two metrics: variances of differences for measuring the smoothness of time series and kurtosis for measuring the preservation of outlyingness and deviations. Our user studies demonstrated that this combination correlates with users' ability to perceive trends and deviations in time series. Compared to displaying raw data, smoothing visualizations using these metrics improves users' accuracy in identifying anomalies by up to 38.4% and decreases response times by up to 44.3%.

However, an exhaustive search for smoothing parameters using the above metrics requires over an hour for 1 million points, in contrast to the sub-second response times required by our applications. To address the second question of scalability, ASAP introduces several optimizations that combine traditional query optimization techniques with visualization properties, such as pushing down constraints from the resolution of display devices to reduce the search space of the smoothing parameters. ASAP is implemented as a time series explanation operator in the MacroBase anomaly detection and explanation engine [29], and as an open-source JavaScript library[1]. The resulting ASAP prototypes demonstrate order-of-magnitude runtime improvements over alternative search strategies while producing high-quality smoothed visualizations.

FASTer: End-to-End Earthquake Detection. FASTer[2] is an unsupervised, end-to-end earthquake detection system built in collaboration with geologists at Stanford University. Based on the high waveform similarity between reoccurring earthquakes, FASTer identifies potential earthquakes by performing a pairwise similarity search on ground motion measurements collected from seismometers. A prototype implementation based on locality-sensitive hashing (LSH) already shows promises of enabling discoveries of low-magnitude earthquakes [108].

However, geologists have encountered two bottlenecks in scaling this analysis beyond three months of continuous data collected at a single seismic station. Contrary to what theory suggests, the actual runtime of LSH in FASTer grows almost quadratically with the data size. One reason for this poor scaling is that seismic data is highly noisy and contains many similar time series patterns generated from repeating background noises. These large numbers of similar, non-seismic patterns also crippled seismologists' ability to manually sift through the output, which can number in the billions of events. To address the computation bottleneck, FASTer leverages domain knowledge,

[1] Demo and code available at `http://futuredata.stanford.edu/ASAP`
[2] Tutorial and code available at `https://github.com/stanford-futuredata/FAST`

		Target Bottleneck	
Chapter	Task	Computation	Human
3	Approximate Aggregate SQL Queries	✓	
4	Approximate Kernel Density Estimation	✓	
5	Time Series Visualization		✓
6	End-to-End Earthquake Detection	✓	✓

Table 1.1: Dissertation Overview

such as the occurrence frequency of earthquakes, to prune noise-related events early in the processing and accelerate the similarity search computation. To address the attention bottleneck, FASTer aggregates raw similarity search outputs across multiple seismic stations to improve the confidence of detected events and reduce the volume to be manually inspected.

Using algorithmic design, optimization based on domain knowledge, and data engineering, we scale the earthquake analysis to ten years of continuous data collected from multiple stations. Compared to supervised methods, FASTer has a much lower false-positive rate, which saves analysts time in result validation. This overall improved scalability has enabled new scientific discoveries, including the detection of previously unknown earthquakes in New Zealand and near a nuclear power plant in California.

1.4 Organization

In this dissertation, we develop systems and algorithms that efficiently use the limited computational resources and analysts' time to scale up the data volumes being processed. Each piece of work addresses one or more processing bottlenecks for specific analytics tasks (Table 1.1). The dissertation is organized as follows:

1. Chapter 2 provides background and describes related work in visual analytics systems as well as the approximation and hashing techniques that our work draws upon.

2. Chapter 3 presents PS^3, the first AQP system to support weighted samples over data partitions. PS^3 leverages precomputed summary statistics to better sample data partitions at query time. PS^3 offers up to a 70× reduction in the query execution time to achieve the same error as uniform partition sampling, with a per-partition storage overhead of less than 100 KB.

3. Chapter 4 describes HBE, an AQP system for approximating kernel density estimation in high dimensions. HBE precomputes a specialized hash index to better sample data points that contribute significantly to the density at query time. With several improvements that reduce the precomputation and sampling overheads, HBE performs up to 10× faster at query time on real-world datasets than the second-best method with comparable precomputation costs.

4. Chapter 5 introduces ASAP, a new analytics operator that produces summary visualizations that highlight large-scale trends and deviations in time series. Specifically, ASAP automatically smooths streaming time series by adaptively optimizing the trade-off between noise reduction and trend retention and is optimized to execute at scale. We demonstrate that ASAP can improve users' accuracy in identifying long-term deviations in time series by up to 38.4% while reducing response times by up to 44.3%.

5. Chapter 6 presents a case study scaling FASTer, an end-to-end earthquake detection system created in collaboration with geologists at Stanford. Incorporating domain knowledge of the earthquake events helps address the processing bottlenecks related to computational resources and manual inspections of results. Our optimizations resulted in an over 100× end-to-end speedup and have enabled discoveries of new earthquake events.

6. Chapter 7 discusses the limitations of the proposed approach and concludes by outlining directions for future work.

The contents of Chapter 3 are adapted from our VLDB'20 paper [268]. The contents of Chapter 4 are adapted from our ICML'19 paper [295]. This work was done in collaboration with PhD student Paris Siminelakis. I contributed to the development of the hashing-based sketch and the cost-based optimizer, and I was responsible for the experimental evaluation. The contents of Chapter 5 are adapted from our VLDB'17 paper [266]. The contents of Chapter 6 are adapted from our VLDB'18 paper [269]. This work was done in collaboration with PhD students Clara Yoon (Geophysics) and Karianne Bergen (Institute for Computational and Mathematical Engineering). I was responsible for the implementation and optimization of the end-to-end system and the experimental evaluation.

The contributions of this dissertation have also had real-life usages and deployments. In particular, ASAP has been incorporated into the open-source monitoring tool Graphite [341], TimescaleDB Toolkit [23], and a JavaScript library downsample [231]. ASAP has also directly inspired an auto smoother for real-time dashboards at Datadog [26]. In addition, the improved scalability and usability of FASTer has led to the discovery of hundreds of new earthquake events and a publication in the seismology journal Bulletin of the Seismological Society of America [338]. FASTer is open-source and has received interest from researchers globally, including McGill University, University College London, the Dublin Institute for Advanced Studies, Austral University of Chile, and the National Geophysical Research Institute in India.

Chapter 2

Background and Related Work

This chapter provides background context and reviews related work on relevant approximation and hashing techniques and visualization systems. Section 2.1 reviews the precomputation costs and query performance trade-offs in existing AQP systems. Section 2.2 reviews locality-sensitive hashing, a hashing technique that we repeatedly used during precomputation to improve the computational efficiency of specific analytics tasks. Section 2.3 reviews related work on visualization recommendation systems, which share our design goals of improving human efficiency in identifying important behaviors in data.

2.1 Approximate Query Processing

AQP systems allow users to trade off between accuracy and query execution time. In applications such as interactive data exploration and visualization, this trade-off is not only acceptable but often desirable: many studies have shown that shorter response times can lead to increased user productivity in exploratory analyses [55, 208]. The ability to provide approximate answers to queries at a fraction of the actual cost enables analysts to explore large datasets efficiently, interact with visualizations in real-time, test hypotheses quickly and identify problematic subsets of the data for drill-down analyses.

AQP techniques focus on aggregate queries of the form: `SELECT` *agg* `FROM` table `WHERE` condition `GROUP BY` dimensions. The query computes a `SUM`-like aggregation over user-defined functions involving one or more numeric columns, along with optional predicates and group-by operators. These queries are common and important in online analytical processing (OLAP) workloads and business intelligence and decision support applications [71].

Existing AQP techniques can be broadly classified into two categories: sampling-based and precomputation-based [202]. These techniques offer different trade-offs in precomputation costs and query performance gains, which we discuss in greater detail below.

2.1.1 Sampling-based AQP

Sampling-based techniques compute a dataset sample when a query arrives and use the sample to estimate query answers. Taking a uniform sample of the dataset during query processing is always possible and requires no prior knowledge of the query workload and no precomputation.

Confidence intervals (CIs) are often used to communicate result quality to users for sampling-based methods. For example, a 95% CI of $x \pm \alpha$ means that the true value of the query answers is within $\pm\alpha$ of the current estimate x with 95% probability. Depending on assumptions on the data distribution and the form of the queries, CIs can be computed using closed-form estimations based on the central limit theorem or tail bounds such as Hoeffding's inequality [157], or using simulation-based methods such as bootstrap [109]. The former requires manual derivations and only applies to relatively simple queries, while the former can handle arbitrarily complex queries, albeit at a much higher computation cost [7]. Although CIs are widely used as error measures, recent studies have shown that visualizations with CIs can cause users to underestimate the variability of results [158]. Therefore, many opportunities remain in designing interfaces to communicate error and uncertainty to users in AQP systems effectively.

Although computation is much more efficient using samples, two factors limit the performance gains enabled by query-time sampling techniques. First, the cost of sample construction during query time is non-trivial. Without any precomputation, query-time sampling techniques often require at least one full pass over the data to generate random samples, limiting the best-case gains. For example, a recent study on a production AQP system at Microsoft shows that sampling only offers significant speedups for complex queries where substantial processing remains after the initial sampling pass [176]. Second, random sampling can lead to large errors in the estimates for rare records and subgroups. For group-by queries with small groups and datasets that exhibit significant value skew, random sampling may miss the small groups and records with large values entirely. Since skew in data distribution and query results are expected in practice, performance improvements enabled by random sampling can be limited.

The following sections discuss two lines of work that collect random samples at query time.

Query-time Sampling. One line of work focuses on developing efficient uniform sampling operators that expose the performance and quality trade-off to users via a parameter that specifies the desired sampling fraction. Early work in the development of such operators started in the 1990s [237], and uniform sampling operators are supported in most databases and big-data systems today.

While it is relatively straightforward to support uniform sampling for a simple aggregation query over the entire dataset, designing sampling operators for more complex queries with predicates, group-bys, and joins requires additional consideration. To support many-to-many joins, for example, we would like to take a random sample over the join results of the two tables (join-then-sample) instead of sampling each table independently at random before joining (sample-then-join). However, the former is much more expensive since it involves a full join. Many efforts have been made

in designing new algorithms and query optimization techniques that support pushing down the sampling operator past group-bys and joins [73,179,340].

Online Aggregation. Online aggregation (OLA) also collects uniform samples of the dataset at query time [60,142,151,240]. Instead of asking users to specify a sampling fraction upfront, in OLA, query estimates are progressively refined as more samples are processed, and the sample sizes can be determined dynamically according to desired result quality (e.g., measured via confidence intervals) or time constraints.

One caveat for OLA methods is that they typically assume that records are processed in a random order or that each prefix of the input is a uniform sample of the dataset. Otherwise, progressively generating random samples would incur high I/O costs from repeat random accesses over the data. The prefix requirement means that the data platforms need to either require random shuffling as an expensive preprocessing step [54], or actively maintain random samples, which can be challenging under insertions and deletions [128,169,232]. Therefore, OLA techniques require more work to integrate into existing data platforms compared to query-time sampling techniques.

2.1.2 Precomputation-based AQP

Precomputation-based techniques compute compact summaries of the dataset based on prior knowledge of the workload and combine precomputed results to answer queries approximately at query time. Precomputation takes a variety of forms, ranging from specialized sketches designed for particular queries to generic summaries such as precomputed samples.

A related idea to precomputation is materialized view. A materialized view contains precomputed query results for a small set of important queries [145,149]. The main difference is that materialized views provide exact answers while AQP techniques provide approximate answers. The discussions below focus on precomputation techniques for AQP.

Sketches. Precomputed sketches are compact summaries of the dataset, usually designed to support specific aggregate queries [82]. Sketches have had much influence from the streaming processing community [229], which focuses on settings where there is not enough space to retain all data, and data must be processed on-the-fly as it arrives. As a result, sketches are usually compact and can be computed in one pass over the dataset. The sizes of sketches are determined by the desired error guarantee of the approximation, a parameter that users can tune based on their needs. Once computed and stored, the sketches can be used to answer queries approximately without needing access to raw data, leading to significant query performance improvements.

We describe a few common types of sketches below. See [85] for an in-depth survey on this topic.

- Distinct value sketch helps answer `COUNT` and `COUNT DISTINCT` queries such as estimating the number of distinct IP addresses visiting a website. Examples of distinct value sketch include the HyperLogLog sketch [117] and the KMV (k minimum values) sketch [32].

- Heavy hitter sketch helps track "hot" items and their frequencies to answer queries such as finding top keywords in recent Google search queries. A class of heavy hitter sketches uses deterministic algorithms based on counters (e.g., MG sketch [225] and SpaceSaving sketch [223]), while others use randomized algorithms based on hashing techniques (e.g., Count sketch [62] and Count-Min sketch [84]).

- Quantile sketch helps answer queries such as computing the 99th percentile latency for the web application. Some quantile sketches assume that inputs are drawn from a fixed domain (e.g., Q-Digest [290]), while others are purely based on comparisons and relative ordering of items (e.g., the GK sketch [139] and the KLL sketch [181]).

While these sketches can enable significant improvements in query performance, they only support limited types of aggregate functions. It is also non-trivial to support queries with predicates and group-bys. Taking the heavy-hitter query that tracks top keywords in Google queries as an example. If we were to query for top keywords in different regions in the states and over different time windows, we would need to partition the dataset by time and geographic coordinates, and store one sketch for each combination. During query time, sketches for relevant data partitions can be merged to answer queries with predicates over location and time. It is easy to see that the total space required to store these sketches grows exponentially as we increase the number of dimensions in queries. Therefore, large compute and storage overheads are required to enable slicing and dicing along different dimensions in a dataset.

Samples. In addition to specialized sketches, precomputation can also take the form of "generic" summaries such as samples. Based on knowledge of the data distribution and the query workload, AQP systems can precompute and store a biased sample of the dataset, which is used to produce approximate query answers during query time. Compared to taking uniform samples at query time, precomputed samples avoid full scans of the original dataset, offering faster query response time. Compared to uniform samples, precomputed samples can also provide more accurate results for outliers, skewed datasets, and rare groups in group-by queries.

Many techniques have been proposed to improve the quality of precomputed samples, such as using stratified sampling techniques to allocate appropriate sampling fractions to different subsets of the dataset. One line of work improves precomputation using information about the dataset. For example, outlier indexes [67] and small group sampling [27] augment uniform samples with a small number of samples with outlying values or from rare groups; congressional sampling optimizes sampling fractions across all possible groups in group-by queries to maximize the accuracy [3, 5]. Other works have leveraged information about the query workload. For example, BlinkDB abstracts workload information into sets of columns that appear in a query (called QCSs) and allocates the sampling budget across different QCSs instead [9].

However, a known problem with both materialized views and precomputed samples is the large

storage overheads. Prior work has shown that it is challenging to construct a sample pool that offers good results for arbitrary queries given feasible storage budgets [179]. For precomputation techniques that leverage workload information, maintaining the precomputed samples under shifts in workload distributions can also be an additional expensive recurring cost.

In summary, existing AQP techniques demonstrate an unsatisfactory trade-off between small performance improvements with random sampling or large improvements with large precomputation overheads. Instead, we seek to combine the best of both worlds by leveraging precomputation not to estimate query answers directly but to provide additional information to improve sampling decisions at query time. A recent system AQP++ [247] also proposed to connect precomputation- and sampling-based methods, albeit via a different approach of using samples to estimate the difference between actual and precomputed query answers. Our approach is similar to the idea of using auxiliary data structures to augment random sampling (e.g., outlier index [67]), except that our approach performs non-uniform, importance-style sampling at query time. In Chapters 3 and 4, we introduce two AQP systems that leverage precomputed synopses and hashing indexes to improve the sampling efficiency at query time.

2.2 Locality-Sensitive Hashing

LSH is a well-known probabilistic method for dimensionality reduction. LSH hashes similar high-dimensional data to the same hash bucket in the low-dimensional space with high probability.

LSH was first introduced as a new technique for approximate nearest neighbor search in high dimensions [168], which have numerous applications in fields such as information retrieval [97], recommendation systems [275], image and video databases [113], and machine learning [14,88,116]. Prior to LSH, indexing techniques for similarity searcher were largely based on tree-style space partitioning methods [37,115,282], which degrade to linear searches for sufficiently high dimensions [326]. At around the same time, motivated by applications to cluster documents and deduplicate webpages, Broder et al. introduced min-wise independent permutations to approximate set similarities [51–53]. Min-wise independent permutations are, in fact, a special instance of an LSH scheme. More formally, an LSH family is defined as follows:

Definition. Let $\mathcal{F}$ be a family of hash functions defined for a metric space $\mathcal{M} = (M, d)$. A hash function in the family $f : \mathcal{M} \to S$ maps elements from the metric space to hash buckets $s \in S$. For any two points $x, y \in M$ and any hash function $f \in \mathcal{F}$, $\mathcal{F}$ is said to be (d_1, d_2, p_1, p_2)-sensitive for parameters $d_1 < d_2$, $p_1 > p_2$:

1. If $d(x, y) \leq d_1$, then $f(x) = f(y)$ with probability at least p_1.
2. If $d(x, y) \geq d_2$, then $f(x) = f(y)$ with probability at most p_2.

Intuitively, this definition ensures that similar items are much more likely to get hashed to the same hash bucket than dissimilar items. LSH, therefore, allows the similarity search to focus on a small set of nearby items instead of searching the entire dataset. To locate a close neighbor of a query item, it often suffices to check only the items that fall into the same hash bucket as the query.

Classic LSH algorithms select hash functions from a distribution independent from the data. A more recent line of work aims to improve the efficiency of classic LSH algorithms by constructing the hashing scheme in a data-dependent manner [19]. While data-dependent LSH shows improvements over classic LSH in Euclidean and Hamming spaces [20], the former do not naturally support insertion and deletion operations, unlike the classic LSH algorithms. In this dissertation, we applied existing LSH techniques and did not contribute to the development of new hashing schemes.

2.2.1 LSH for Nearest Neighbor Search

The nearest neighbor search problem is the following: given a set of points P in some metric space $\mathcal{M}$ and a query point $q \in \mathcal{M}$, find the point in P that is closest to q. For example, $\mathcal{M}$ can be the d-dimensional Euclidean space $\mathbb{R}^d$ under Euclidean distance (l_2 norm) or Manhattan distance (l_1 norm). Two variants of this problem have also been widely studied: k-nearest neighbor (k-NN) search and ϵ-approximate nearest neighbor search. The k-NN search finds not only one, but k closest points to the query point. The ϵ-approximate nearest neighbor search finds all points that are within a distance of $(1+\epsilon)R$ to the query point, where R defines the distance between the query point and its true nearest neighbor. We focus the discussion below on the approximate nearest neighbor (ANN) search problem.

Many algorithms have been proposed to find approximate nearest neighbors, most notably graph-based and hashing-based methods. Graph-based methods construct a nearest neighbor graph from the dataset, where each node corresponds to a data point and nodes are connected to its nearest neighbors via directed edges [103,144]. Most graph-based methods process ANN queries by making greedy traversal steps on the graph towards the query point [150,214,215]. Graph-based methods have shown empirically promising results in recent performance benchmarks [25], but many open questions remain regarding the theoretical guarantees that they can achieve [118,194,257]. Hashing-based methods, on the other hand, project data points into a low dimensional representation and construct an index by storing data that share the same representation into a hash bucket[1]. As explained above, the locality-sensitive property guarantees that similar items are more likely hashed to the same hash bucket than dissimilar items, making the index an efficient data structure for ANN search. Many locality sensitive hashing schemes have been developed for a range of similarity measures (Table 2.1). LSH remains one of the most widely used techniques for approximate similarity search in the database and data mining communities [90,133,216,259].

[1] We omit other hashing techniques for ANN search here, such as using hash codes to approximate the distance between data points. [324] provides a survey on this topic.

Hash Function	Similarity Measure
MinHash [51]	Jaccard similarity
Original LSH [168]	Hamming distance
SimHash [65]	Cosine similarity
E2LSH [93]	Euclidean distance

Table 2.1: Example LSH hashing schemes and similarity measures

LSH-based nearest neighbor search algorithms generally work as follows. In the precomputation phase, we construct a few independent hash tables from the dataset. In the query phase, we consider a data point a nearest neighbor candidate if it lands in the same hash bucket as the query point in *any* of hash tables. We compute the actual distances between the query point and retrieved candidates and return ones that satisfy the distance threshold. To search large datasets efficiently, we hope that dissimilar items will not get hashed to the same hash bucket (low false positives), so that most of the items we retrieve are true near neighbors; to get high-quality results, we hope that similar items get hashed to at least one of the hash tables (low false negatives), so that our candidate set covers most of the query item's true nearest neighbors. Users can tune the quality of LSH-based ANN search via two main parameters (k, l):

- k controls the precision of the search. Given a (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{F}$, it is possible to a new (d_1, d_2, p_1^k, p_2^k)-sensitive family $\mathcal{G}$ by concatenating k independently chosen hash functions $f_1, ..., f_k$ from $\mathcal{F}$. For each hash function g in the new family, $g(x) = g(y)$ if and only if $f_i(x) = f_i(y)$ for all $i = 1, 2, ..., k$. Large values of k reduces the false positives by requiring nearest neighbor candidates to match all k hash values simultaneously.

- l controls the recall of the search. We construct l independent hash tables, each of which uses hash functions $g_1, g_2, ..., g_l$ that are chosen independently and uniformly at random from $\mathcal{G}$. Given a query item q, the algorithm collects the contents of the hash bucket that the query falls into from each table $g_i(q), ..., g_l(q)$, and returns the union of the l buckets as the candidate set. Large values of l reduces the false negatives by increasing the total number of hash buckets to be checked, at the cost of increased precomputation cost.

We illustrate the effect of these parameters with a concrete example using MinHash [51]. For two d-dimensional binary vectors $x, y \in \{0, 1\}^d$, MinHash guarantees that the probability x, y are hashed to the same bucket (also called collision probability) equals their Jaccard similarity: $J(x, y) = \frac{|x \cap y|}{|x \cup y|}$. For two items with collision probability p, concatenating k hash functions effectively decreases this probability to p^k, and using l hash tables increases the probability that the two items become nearest neighbor candidates with probability $1 - (1 - p^k)^l$. This function takes the form of an *S-curve*, regardless of choices of k and l (Figure 2.1).

Chapter 6 presents a novel application of LSH-based ANN search in earthquake detection. The application leverages the property that reoccurring earthquakes have near-identical waveforms and

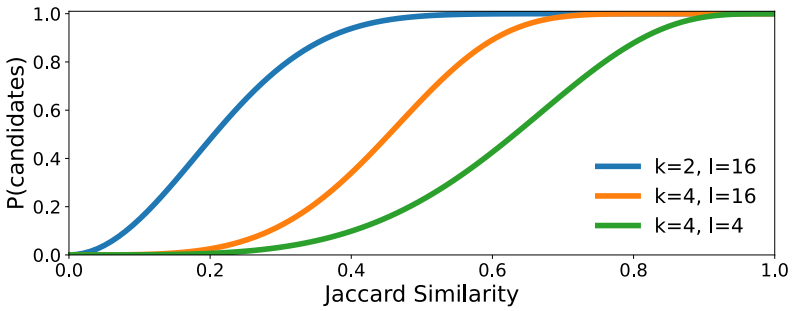


Figure 2.1: The probability that two items become nearest neighbor candidates under a varying number of hash functions k and hash tables l using MinHash. The probability takes the form of an *S-curve*, regardless of choices of k and l.

performs an all-pair similarity search on seismic data using MinHash LSH. However, we found that an off-the-shelf implementation of LSH was often proved insufficient, and that domain-specific optimizations were necessary to enable LSH performances to meet the requirements of our analytics tasks. For example, the original Minhash implementation showed near quadratic scaling of query time with the dataset size due to repeating noises and correlations in seismic signals. We leveraged domain-specific filters and constraints to prune large hash buckets containing repeating background noise, which significantly improved the search efficiency. We also observed that LSH parameters with similarity success probability curves can have vastly different query performance in practice. In fact, automatic tuning of LSH parameters remains an active area of research [33, 104]. We believe that the challenges and opportunities of adapting class LSH algorithms into a specific domain offer valuable lessons for future researchers, such as the importance of incorporating domain knowledge and handling of data skew.

2.2.2 LSH for Sampling

While LSH has been most commonly associated with ANN search, a recent line of work explores the idea of using LSH as a sampler. On the high level, the idea works as follows. During the precomputation phase, we construct an index on the dataset using appropriate LSH families and parameters. During the query phase, we sample items from hash buckets that the query item falls into, and weigh the samples appropriately according to the collision probability.

To see how LSH-based sampling could help, we first introduce the *importance sampling* technique. Suppose that we want to estimate the mean of a function f over distribution X. Therefore is an event E for which $\mathbb{P}(E)$ is small, but $f(x)$ is close to zero outside E. A simple uniform sample over the distribution X is unlikely to contain any point inside E and therefore, leads to large errors in the estimate. Ideally, we would like to get more samples in E, and we can achieve so by obtaining samples from a different distribution that over-samples the important regions/events. Importance

sampling refers to this technique for estimating properties (e.g., mean) of a particular distribution with samples generated from a different distribution. Concretely, suppose that our goal is to estimate $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$. The quality of this estimate depends on the variance of the data distribution. We have a separate sampling distribution Q over $[n]$ such that $\mathbb{P}(Q = i) = q_i$. An unbiased estimator for μ is given by sampling $I \in [n]$ according to Q and averaging importance weights $\frac{w_I}{q_I}$ across the samples. We can reduce the variance of the estimator by setting Q appropriately, and the minimum variance is obtained by setting $q_i^* = \frac{x_i}{\sum_j x_j}$, or sampling data points with probability proportional to their contribution to the final estimate.

The key insight behind this sampling view of LSH is to take advantage of the fact that the collision probability is defined precisely in LSH families. Therefore, we can manipulate the collision probability to construct our desired sampling distribution by choosing the appropriate hash families and search parameters (k, l). For example, [63] shows that with specific choices of k, Euclidean LSH [93] can create collision probabilities appropriate for approximating kernel density estimates with Exponential and Gaussian kernels. Similarly, [300] shows that MIPS LSH [288], with appropriate choices of k, l, creates good sampling distributions for approximating partition functions in log-linear models. This idea has been used in several applications, such as in improving the training and inference efficiency of large neural networks [74, 189, 301].

Chapter 4 leverages LSH-enabled importance sampling to approximate KDEs. This is a direct follow-up work of Hashing-Based-Estimator [63], which first demonstrated provable improvements over random sampling for the approximation problem in high dimensions. Our focus, however, is on the practical challenges that prevent the theoretical advances in sampling efficiency from translating into actual speedups in query performance. For example, HBE's precomputation phase requires constructing many hash tables over the entire dataset, the cost of which can outweigh gains in the downstream sampling efficiency. To address this challenge, we introduced a sketch that significantly reduces hash table sizes while preserving the properties of the hashing step. By improving several aspects of the HBE, such as the precomputation overheads described above, we demonstrate for the first time that this hashing-based approach can outperform random sampling in query performance on real-world datasets.

2.3 Visualization Systems

Information visualization is "the use of computer-supported, interactive, visual representations of abstract data in order to amplify cognition" [57]. Visualization allows a large amount of data to be represented in a limited space [313], and enhances the recognition of patterns through abstraction and the selective omission of information [264]. Given the large volumes of data generated today, data analysts increasingly rely on visualization tools such as Excel and Tableau to assist in data understanding and decision making.

To help amplify cognition, prior works have explored methods to automatically generate and recommend visual representations of the data, which can be particularly useful for use cases such as exploratory analyses [314]. For example, Mackinlay's pioneer work APT [212] enumerates the visualization space via a composition algebra and prunes and ranks visualizations based on a set of expressiveness and effective criteria. Tableau's Show Me [213] demonstrates that setting good default configurations in visualizations improves the user experiences of both new and experienced users. Building upon this line of work, visualization systems that automatically generate and recommend visualizations to users have recently become a topic of active interest in the database and human-computer interaction communities [227, 316, 329, 330].

Two common challenges of designing and implementing visual recommendation systems are scalability and utility. First, the system must search through a large number of candidates (e.g., tens of thousands of visualizations in ZenVisage [294]) while being able to respond at interactive speeds. Recent systems such as SeeDB [317] have leveraged query optimization inspired techniques to prune low-utility visualizations and share computation from multiple queries to enable scalable visualization recommendations. Second, the system must quantitatively assess the interestingness of visualizations, which can vary across different use cases. For example, Profiler [175] is a visual analysis tool designed specifically for discovering data anomalies. Therefore, its ranking metric is based on the mutual information between detected anomalies and data columns. In contrast, Voyager [329] is a general recommendation-powered visualization browser that does not have knowledge of users' tasks. Therefore, it aims to provide a diverse set of visualizations for users to examine ranked by the perceptual effectiveness metrics proposed by early work [80, 212].

Chapter 5 introduces ASAP, a visualization operator designed to recommend smoothing parameters for time series visualizations in monitoring dashboards that can help hide local fluctuations and highlight large-scale trends and deviations. Similar to Show Me, ASAP tries to improve user experiences by setting smart defaults. The fast-arriving nature of data streams combined with the need to identify abnormal behaviors require a new solution to address the unique scalability and utility challenges in monitoring applications. ASAP leverages both resolution constraints from the display device and statistical properties of the dataset to prune the search space and enable interactive rendering for streaming data. ASAP also introduces novel metrics to quantify the quality of the smoothed visualizations in highlighting large-scale deviations, the effectiveness of which are validated via user studies.

Part I

Improving Computational Efficiency

Chapter 3

PS^3: Approximate Query Processing with Partition Samples

This chapter presents PS^3, the first AQP system to support non-uniform samples over data partitions via the novel use of precomputed partition-level summary statistics. Much of the existing AQP literature focuses on row-level samples [5,27,67], but constructing a row-level sample can be expensive when data is stored in media that does not support random access (e.g., flat files in data lakes and columnar stores [99,293]). For example, if data is split into partitions with 100 rows, a 1% uniform row sample would in expectation require fetching 64% ($1-0.99^{100}$) of the partitions; a 10% uniform row sample would touch almost all partitions. Recent work from a production AQP system has shown that row-level sampling only offers significant speedups for complex queries where substantial query processing remains after the sampling [177].

In contrast to row-level sampling, the I/O cost of constructing a *partition-level* sample is proportional to the sampling fraction[1]. In the example above, a 1% partition-level sample would only read 1% of the data. We are especially interested in big data clusters, where data is stored in chunks of tens to hundreds of megabytes rather than in disk blocks or pages which are typically a few kilobytes [130,293]. Partition-level sampling is already used in production due to its appealing performance: commercial databases create statistics using partition samples [91,239] and several Big Data stores allow sampling partitions of tables [47,253,299].

However, a key challenge remains in how to construct partition-level samples that can answer a given query accurately. Since all or none of the rows in a partition are included in the sample, the correlation between rows (e.g., due to layout) can produce inaccurate answers. A uniformly random partition-level sample is not representative of the dataset unless the rows are randomly distributed among the partitions [69], which rarely occurs in practice [54]. In addition, even a uniform random

[1] In this chapter, we use the term "partition" to refer to the finest granularity at which the storage layer maintains summary statistics.

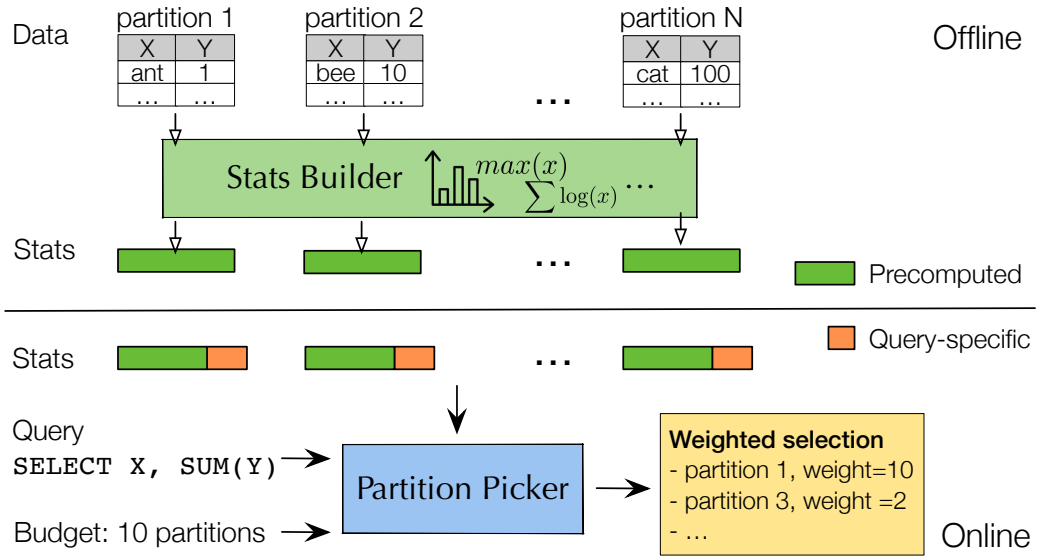


Figure 3.1: Our system PS^3 makes novel use of summary statistics to perform importance and similarity-aware sampling of partitions.

sample of rows can miss rare groups in the answer or miss the rows that contribute substantially to `SUM`-like aggregates. It is not known how to compute non-uniform samples (e.g,. [5, 101]) over partitions, which would help with queries with group-bys and complex aggregates.

Our system PS^3 (Partition Selection with Summary Statistics) supports AQP via weighted partition selection (Figure 3.1). Our primary use case is in large-scale production query processing systems such as Spark [22], F1 [292], and SCOPE [58] where queries are read-only and datasets are bulk appended. Our goal is to minimize the approximation error given a sampling budget, or fraction of data that can be read. Motivated by observations from production clusters at Microsoft and in literature that many datasets remain in the order that they were ingested [178], PS^3 does not require any specific layout or re-partitioning of data. Instead of storing precomputed samples [9, 27, 68], which requires significant storage budgets to offer good approximations for a wide range of queries [71, 179], PS^3 performs sampling exclusively during query optimization. Finally, similar to the query scope studied in prior work [9, 242, 305], PS^3 supports single-table queries with `SUM, COUNT(*), AVG` aggregates, `GROUP BY` on columnsets with moderate distinctiveness, predicates that are conjunctions, disjunctions or negations over single-column clauses.

PS^3 novelly combines a set of inexpensive, precomputed partition-level summary statistics to help select partitions that are most relevant to a query. The key question is which statistics to use. Systems such as Spark [255] and Oracle Database [238] already maintain statistics such as the maximum and minimum values of a column to assist in query optimization [178]. Following similar design considerations, we look for statistics with small space requirements that can be computed

for each partition in a single pass at ingest time. For functionality, we seek statistics that are discriminative enough to support decisions such as whether the partition contributes disproportionally large values of the aggregates. We propose such a set of statistics for partition sampling – measures, heavy hitters, distinct values, and histograms – that includes but expands on conventional catalog-level statistics. The total storage overhead scales with the number of partitions instead of with the dataset size. We only maintain single-column statistics to keep the overhead low. While the set of statistics is not complete, each type of statistics contributes to the sampling performance, and, in aggregate, delivers effective AQP results.

Next, we propose three ways of using precomputed summary statistics to help improve partition selection during query time:

- To improve the approximation quality, we wish to sample partitions that contribute significantly to the query answer more frequently. While it is challenging to manually design rules that relate summary statistics to partition contribution, a model can learn to what degree summary statistics matter through examples. Inspired by prior work that uses learning techniques to improve row-level sampling [321], we propose a learned importance-style sampling algorithm that works on aggregate queries with `GROUP BY` clauses and on partitions. The summary statistics serve as a natural feature representation for partitions, from which we can train models offline to learn a mapping from the summary statistics to the relative importance of a partition. During query optimization, the trained models classify partitions into several importance groups and allocate the sampling budget across groups such that more important groups receive more samples. The training overhead is a one-time cost for each dataset and workload, and for datasets that are frequently queried, this overhead is amortized over time.

- For two partitions that output similar answers to an input query, it suffices to only include one of them in the sample. While directly comparing the contents of the two partitions is expensive, the query-specific summary statistics can act as a proxy for the similarity between partitions. Specifically, we cluster data partitions according to normalized versions of their summary statistics and select one sample from each cluster to minimize sample redundancy.

- Datasets commonly exhibit significant skew in practice. For example, in a prototypical production service request log dataset at Microsoft, the most popular application version out of 167 distinct versions accounts for almost half of the dataset. Inspired by prior works in AQP that recognize the importance of outliers [27, 67], we use summary statistics (e.g., occurrences of heavy hitters in a partition) to identify a small number of partitions that are likely to contain rare groups and dedicate a portion of the sampling budget to evaluate these partitions exactly.

In this chapter, we make the following contributions:

1. We introduce PS^3, a system that makes novel uses of summary statistics to perform weighted partition selection for many popular queries. Given the query semantics, summary statistics,

and a sampling budget, the system intelligently combines a few sampling techniques to produce a set of partitions to sample and the weight of each partition.

2. We propose a set of lightweight sketches for data partitions that are both practical to implement and can produce rich partition summary statistics. While the sketches are well known, this is the first time the statistics have been used for weighted partition selection.

3. We evaluate on several real-world datasets with real and synthetic workloads. Our evaluation shows that each component of PS^3 contributes meaningfully to the final accuracy and that together, the system outperforms alternatives across datasets and layouts, delivering a 2.7× to 70× reduction in data read given the same error compared to uniform partition sampling.

The remainder of this chapter proceeds as follows. Section 3.1 provides an overview of PS^3's architecture and major design considerations. We describe the precomputation phase of PS^3 in Section 3.2 and the query-time sampling phase in Section 3.3. Section 3.4 evaluates PS^3's empirical performances across datasets, data layouts, and different parameter settings. Section 3.5 discusses related work and concludes with directions for future work.

3.1 System Overview

In this section, we give an overview of PS^3, including its major design considerations, supported queries, inputs, and outputs, and the problem statement.

3.1.1 Design Considerations

We highlight a few design considerations in the system.

Layout Agnostic. A random data layout would make the partition selection problem trivial, but maintaining a random layout requires additional efforts and rarely happens in practice [54]. In read-only or append-only data stores, it is also expensive to modify the data layout. As a result, we observe that in practice, many datasets simply remain in the order that they were ingested in the cluster. In addition, prior work [305] has shown that it is challenging and sometimes impossible to find a partitioning scheme that enables good data skipping for arbitrary input queries. Therefore, instead of requiring re-partitioning or random layout, PS^3 explicitly chooses to keep data in situ and tries to make the best out of the given data layout. We show that PS^3 can work across different data layouts in the evaluation (§ 3.4.5).

Sampling on a single table. To perform joins effectively, prior work [179] has shown that sampling each input relation independently is not enough and that the joint distribution must be taken into account. Handling the correlations between join tables at the partition level is another research problem on its own, and is outside the scope of this paper. However, sampling on a

single table can still offer non-trivial performance improvements even for queries that involve joining multiple tables. For example, in key–foreign key joins, fact tables are often much larger compared to dimension tables. Sampling the fact table, therefore, already gets us most of the gains.

Generalization. Prior works make various trade-offs between the efficiency and the generality of the queries that they support, ranging from having access to the entire workload [305] to being workload agnostic [152]. Our system falls in the middle of the spectrum, where we make assumptions about the structure and distribution of the query workload. Specifically, we assume that the set of columns used in `GROUP BY`s and the aggregate functions are known apriori, with the scope defined in § 3.1.2; predicates can take any form that fits under the defined scope and we do not assume we have access to the exact set of predicates used. We assume that a workload consists of queries made of an arbitrary combination of aggregates, group bys and predicates from the scope of interest. PS^3 is trained per data layout and workload, and generalizes to unseen queries sampled from the same distribution as the training workload. Overall, our system is best suited for commonly occurring queries and should be retrained in case of major changes in query workloads such as the introduction of unseen group by columns.

We do not consider generalization to unseen data layouts or datasets and we view broader generalization as an exciting area for future work (§ 3.5.2). Since most summary statistics are computed per column, different datasets might not share any common statistics. Even for the same dataset, the importance of summary statistics can vary across data layouts. For example, the mean of column X can distinguish partitions in a layout where the dataset is sorted by X, but may provide no information in a random layout.

3.1.2 Supported Queries

In this section, we define the scope of queries that PS^3 supports. We support queries with an arbitrary combination of aggregates, predicates and group bys. Although we do not directly support nested queries, many queries can be flattened using intermediate views [146]. Our techniques can also be used directly on the inner queries. Overall, our query scope covers 11 out of 22 queries in the TPC-H workload (Appendix A.1, extended report [174]).

- `Aggregates`. We support `SUM` and `COUNT(*)` (hence `AVG`) aggregates on columns as well as simple linear projections of columns in the select clause. The projections include simple arithmetic operations (`+, -`) on one or more columns in the table[2]. We also support a subset of aggregates with `CASE` conditions that can be rewritten as an aggregate over a predicate.
- `Predicates`. Predicates include conjunctions, disjunctions and negations over the clauses of the form c op v, where c denotes a column, op an operation and v a value. We support equality

[2]We also support the multiply and divide operations in some cases using statistics computed over the logs of the columns.

and inequality comparisons on numerical and date columns, equality check with a value as well as the `IN` operator for string and categorical columns as clauses.

- `Groups`. We support `GROUP BY` clauses on one or more stored attributes[3]. We do not support `GROUP BY` on columns with large cardinality since there is little gain from answering highly distinct queries over samples; one could either hardly perform any sampling without missing groups, or would only care about a limited number of groups with large aggregate values (e.g., `TOP` queries), which is out of the scope of this paper.
- `Joins`. Queries containing key–foreign joins can be supported as queries over the corresponding denormalized table. For simplicity, our discussion in this paper is based on a denormalized table.

In the TPC-H workload, 16 out of the 22 queries can be rewritten on a denormalized table and 11 out of the 16 are supported by our query scope. For the 5 that are not supported, 4 involve group bys on high cardinality columns and 1 involves the `MAX` aggregate. A number of prior work have also studied similar query scopes [9, 242, 305].

3.1.3 Inputs and Outputs

PS^3 consists of two main components: the statistics builder and the partition picker (Figure 3.1). In this section, we give an overview of the inputs and outputs of each component during preprocessing and query time.

Statistics Builder

Preparation. The statistics builder takes a data partition as input and outputs a number of lightweight sketches for each partition. The sketches are stored separately from the partitions. We describe the sketches used in detail, including the time and space complexity for constructing and storing the sketches in § 3.2.1.

Query Time. During query optimization, one can access the sketches *without* touching the raw data. Given an input query, the statistics builder combines pre-computed column statistics with query-specific statistics computed using the stored sketches and produces a set of summary statistics for each partition and for each column used in the query.

Partition Picker

Preparation. In the preparation phase, the picker takes a specification of workload in the form of a list of aggregate functions and columnsets that are used in the `GROUP BY`. We can sample a query from the workload by combining randomly generated predicates and randomly selected aggregate

[3] To support derived attributes, we make a new column from the derived attribute and store its summary statistics

functions and group by columnsets (0 or 1) from the specification. For each sampled query, we compute the summary statistics as well as the answer to the query on each partition as the training data, which the picker uses to learn the relevance of different summary statistics. The training is a one time cost and we train one model for each workload to be used for all test queries. We elaborate on the design of the picker in § 3.3.

Query Time. The picker takes an input query, summary statistics and a sampling budget as inputs, and outputs a list of partitions to sample, as well as the weight of each partition in the sample. This has a net effect of replacing a table in the query execution plan with a set of weighted partition choices with a small overhead (Table 3.6). Query execution should also be augmented to handle weights, similar to modifications suggested in prior work [179].

3.1.4 Problem Statement

Let N be the total number of partitions and M be the dimension of the summary statistics. For an aggregation query Q, let G be the set of groups in the answer to Q. For each group $g \in G$, denote the aggregate values for the group as $\mathbf{A_g} \in \mathbb{R}^d$, where d is the number of the aggregates. Denote the aggregates for group g on partition i as $\mathbf{A_{g,i}}$.

Given the input query Q, the summary statistics $F \in \mathbb{R}^{N \times M}$ as well as sampling budget n in the form of number of partitions to read, our system returns a set of weighted partition choices $S = \{(p_1, w_1), (p_2, w_2), ..., (p_n, w_n)\}$. The approximate answer $\tilde{\mathbf{A}}_\mathbf{g}$ of group g for Q is computed by $\tilde{\mathbf{A}}_\mathbf{g} = \sum_{j=1}^{n} w_j \mathbf{A_{g,p_j}}, \forall g \in G$.

Our goal is to produce the set of weighted partition choice S such that $\tilde{\mathbf{A}}_\mathbf{g}$ is a good approximation of the true answer $\mathbf{A_g}$ for all groups $g \in G$. To assess the approximation quality across groups and aggregates that are of different sizes and magnitudes, we measure absolute and relative error, as well as the percentage of groups that are missed in the estimate.

3.2 Precomputation: Partition-level Summary Statistics

The high-level insight of our approach is that we want to differentiate partitions based on their contribution to the query answer, and that the contribution can be estimated using a rich set of partition-level summary statistics. As a simple example, for `SUM`-type aggregates, partitions with a higher average value of the aggregate should be preferred, all else being equal. We are unaware of prior work that uses partition-level summary statistics for performing non-uniform partition selection. In this section, we describe the design and implementation of the summary statistics.

Table 3.1: Per partition, the time and space overheads to construct and store sketches for partitions with R_b rows. Small logarithmic factors are ignored.

Sketch	Construction	Storage
Histograms	$O(R_b \log R_b)$	$O(\#buckets)$
Measures	$O(R_b)$	$O(1)$
AKMV	$O(R_b)$	$O(k)$
Heavy Hitter	$O(R_b)$	$O(\frac{1}{support})$

3.2.1 Lightweight Sketches

Our primary use case, similar to columnar databases, is read-only or append-only stores. Summary statistics are constructed for each new data partition when the partition is sealed. The necessary data statistics should be simple, small in size and can be computed incrementally in one pass over data. The necessary statistics should also be discriminative enough to set partitions apart and rich enough to support sampling decisions such as estimating the number of rows that pass the predicate in a partition. We opt to use only single-column statistics to keep the memory overhead light, although more expensive statistics such as multi-column histograms can help estimate selectivity more accurately. The design considerations lead us to the following sketches:

- **Measures:** Minimum, maximum, as well as first and second moments are stored for each numeric column. For columns whose value is always positive, we also store measures on the log transformed column.
- **Histogram:** We construct equal-depth histograms for each column. For string columns, the histogram is built over hashes of the strings. By default, each histogram has 10 buckets.
- **AKMV:** We use an AKMV (K-Minimum Values) sketch to estimate the number of distinct values [45]. The sketch keeps track of the k minimum hashed values of a column and the number of times these values appeared in the partition. We use $k = 128$ by default.
- **Heavy Hitter:** We maintain a dictionary of heavy hitters and their frequencies for each column in the partition using lossy counting [217]. By default, we only track heavy hitters that appear in at least 1% of the rows, so the dictionary has at most 100 items.

Table 3.1 summarizes the time complexity to construct the sketches and the space overhead to store them, ignoring small logarithmic factors. The sketches can be constructed in parallel for each partition. We do not claim that the above choices make a complete set of sketches that should be used for the purpose of partition selection. Our point is that these are a set of inexpensive sketches that can be easily deployed or might have already been maintained in big-data systems [178], and that they can be used in new ways to improve partition sampling.

Table 3.2: Summary statistics and the sketches used to compute them. Selectivity is computed per query and all other statistics is computed per column.

Summary Statistics	**Sketch**
$\overline{x}$, $min(x)$, $max(x)$, $\overline{x^2}$, $std(x)$	Measures
$\overline{\log(x)}$, $\overline{\log(x)^2}$, $min(\log(x))$, $max(\log(x))$	Measures
number of distinct values	AKMV
avg/max/min/sum freq. of distinct values	AKMV
# hh, avg/max freq. of hh	Heavy Hitter
occurrence bitmap of heavy hitters	Heavy Hitter
selectivity	Histogram

3.2.2 Summary Statistics as Features

Given the set of sketches, we compute summary statistics for each partition, which can be used as *feature vectors* to discriminate partitions based on their contribution to the answer of a given query. The features consist of two parts: pre-computed per column features and query-specific selectivity estimates (Table 3.2). We apply a query-dependent mask on the pre-computed column features: features associated with columns that are unused in the query are set to zero. In addition, for categorical columns where the measure based sketches do not apply, we set the corresponding features to zero. The schema of the feature vector is determined entirely by the schema of the table, so queries on the same dataset share the same feature vector schema.

Overall, there are four types of features based on the underlying sketches that generate them: measures, heavy hitters, distinct values and selectivity. Each type of feature captures different information about the partitions and the queries. Measures help identify partitions with disproportionally large values of the aggregates; heavy hitters and distinct values help discriminate partitions from each other and selectivity helps assess the impact of the predicates. We found that all types of features are useful in PS^3 but the relative importance of each varies across datasets (§ 3.4.4).

Extracting features from sketches is overall straightforward; we discuss two special cases below.

Occurrence Bitmap. We found that it is not only helpful to know the number of heavy hitters, but also *which* heavy hitters are present in the partition. To do so, we collect a set of k global heavy hitters for a column by combining the heavy hitters from each partition. For each partition, we compute a bitmap of size k, each bit representing whether the corresponding global heavy hitter is also a heavy hitter in the current partition. The feature is only computed for grouping columns and we cap k at 25 for each column.

Selectivity Estimates. The selectivity estimate is a real number between 0 and 1, designed to reflect the fraction of rows in the partition that satisfies the query predicate. The estimate supports predicates defined in our query scope (§ 3.1.2) and is derived using histograms over individual

columns. Predicate clauses that use the same column (e.g., $X < 1$ or $X > 10$) are evaluated jointly. As a special case, if a string column has a small number of distinct values, all distinct values and their frequencies are stored exactly; this can support regex-style textual filters on the string column (e.g., `'%promo%'`). We use the following four features to represent the selectivity of predicates which can be a conjunction or disjunction of individual clauses:

1. `selectivity_upper`: For `AND`s, the selectivity is at most the min of the selectivity of individual clauses; for `OR`s, the selectivity is at most 1 and at most the sum of the selectivity of individual clauses.

2. `selectivity_indep`: This feature computes the selectivity assuming independence between predicate clauses. For `AND`s, the feature is the product of the selectivity for each individual clause; for `OR`s, the feature is the min of the selectivity of individual clauses.

3. `selectivity_min`, `selectivity_max`: We store the min and max of the selectivity of individual clauses.

If the upper bound of the selectivity is zero, the partition contains no rows that pass the predicate; if the upper bound is nonzero however, the partition can have zero or more rows that pass the predicate. In other words, as a classifier for identifying partitions that satisfy the predicate, `selectivity_upper`> 0 has perfect recall and uncertain precision. For simple predicates such as $X > 1$, the precision is 100%; for complicated predicates involving conjunctions and disjunctions over many clauses and columns (e.g., TPC-H Q19), the precision can be as low as 10%.

3.3 Query-time: Partition Picking

In this section, we describe PS^3's partition picker component and how it makes novel use of the summary statistics discussed above to realize weighted partition selection.

3.3.1 Picker Overview

To start, we give an overview of how our partition picker works. Recall that the picker takes a query, the summary statistics and a sampling budget as inputs, and outputs a list of partitions to evaluate the query on and the weight of each partition. Partial answers from the selected partitions are combined in a weighted manner, as described in § 3.1.4.

Algorithm 1 describes the entire procedure. We first identify outlier partitions with rare groups using the procedure described in § 3.3.4. Each outlier partition has a weight of 1. We then use the trained models to classify the remaining partitions into groups of different importance, using the algorithm described in § 3.3.3. We allocate the remaining sampling budget across groups such that the sampling rate decreases by a factor of α from the i^{th} important to the $(i+1)^{th}$ important group.

Algorithm 1 Partition Picker

Input: partition features F, sampling budget n, group-by columns gb_col, models regrs, decay rate α

Output: selection: $[(p_1, w_1), (p_2, w_2), ..., (p_n, w_n)]$

1: outliers, inliers $\leftarrow$ OUTLIER(F, gb_col)
2: $n_o \leftarrow$ outliers.size()
3: selection.add(outliers, $[1] * n_o$)
4: groups $\leftarrow$ IMPORTANCEGROUP(F, inliers, regrs)
5: $n_c \leftarrow$ ALLOCATESAMPLES(groups, $n - n_o$, α)
6: **for** $i \leftarrow 1, ...,$groups.size() **do**
7: selection.add(CLUSTERING(F[groups[i]], $n_c[i]$))
8: **end for**

Finally, given a sample size and a set of partitions in each importance group, we select samples via clustering using the procedure described in § 3.3.2. An exemplar partition is selected from each cluster, and the weight of the exemplar equals the size of the cluster. We explain each component in detail in the following sections.

3.3.2 Sample via Clustering

We start by describing the sample selection procedure (line 7 in Algorithm 1), designed to leverage the redundancy between partitions. We use feature vectors to compute a similarity score between partitions, which consequently enables us to choose dissimilar partitions as representatives of the dataset. In fact, identical partitions will have identical summary statistics, but the converse does not hold; having summary statistics on multiple columns as well as multiple statistics for each column makes it less likely that dissimilar partitions have identical summary statistics.

We propose to use *clustering* as a sampling strategy: given a sampling budget of n partitions, we perform clustering using feature vectors with a target number of n clusters; an exemplar partition is chosen per cluster, with an assigned weight equals the number of partitions in the cluster. Denote the answer to the query on cluster i's exemplar partition as A_i and the size of cluster i as s_i. The estimate of the query answer is given by $\tilde{A} = \sum_{i=1}^{n} s_i A_i$.

Concretely, we measure partition similarity using Euclidean distances of the feature vectors. We zero out features for unused columns in the query so they have no impact on the result; we also perform normalization such that the distance is not dominated by any single feature (Appendix A.1.1). Regarding the choice of the clustering algorithm, we experimented with KMeans and Agglomerative Clustering and found that they perform similarly. Finally, the cluster exemplar is selected by picking the partition whose feature vector has the smallest distance to the median feature vector of partitions in the clusters.

Our proposed scheme leads to a biased estimator that can be challenging to analyze. Specifically, given the median feature vector of a cluster, our estimator deterministically picks the partition that is

closest to the median vector as the cluster exemplar. However, one could make a simple modification to unbias the estimator by selecting a random partition in the cluster as the exemplar instead. We have included an empirical comparison of the accuracy of the two estimators as well as a variance analysis for the unbiased estimator in Appendix A.2. We have empirically found that the proposed scheme outperforms its unbiased counterpart when the sampling budget is limited.

Clustering effectively leverages the redundancy between partitions, especially in cases when partitions have near identical features. Although there is no guard against an adversary, in practice, having a large and diverse set of summary statistics makes it naturally difficult for dissimilar partitions to be in the same cluster. Clusters play a similar role as strata in stratified sampling. The goal of clustering is to make partitions in the same stratum homogeneous such that the overall sampling variance is reduced. Finally, clustering results vary from query to query: the same partition can be in different clusters for different queries due to the changes in selectivity features and the query-dependent column masks.

Feature Selection. Clustering assumes that all features are equally relevant to partition similarity. To further improve the clustering performance, we perform feature selection via a "leave-one-out" style test. For example, consider a table with columns X, Y and features min, max. We compare the clustering performance on the training set using $\{min(X), max(X), min(Y), max(Y)\}$ as features to that from using only $\{max(X), max(Y)\}$ as features. If the latter gives a smaller error, we subsequently exclude the min feature for all columns from clustering. We greedily remove features until converging to a local optimal, at which point excluding any remaining features would hurt clustering performance. In an outer loop, we repeat the above greedy procedure multiple times, each time starting with a random ordering of the features. Our experiments show that feature selection consistently improves clustering performance across datasets. We provide the pseudo code of the procedure in Appendix A.1.1.

Limitations. We briefly discuss two failure cases for clustering in which PS^3 can fall back to random sampling. First, clustering takes advantage of the redundancy among partitions. In the extreme case when the query groups by the primary key, no two partitions contribute similarly to the query and any downsampling would result in missed groups. As discussed in § 3.1.2, our focus is on queries where such redundancy exists. Second, queries with highly selective predicates might suffer from poor clustering performances. Since most features are computed on the entire partition, the features would no longer be representative of partition similarity if only a few rows satisfy the predicate in each partition. Although we can use the `selectivity_upper` feature as an upper bound for the true selectivity, in practice, we have seen that this upper bound could overestimated the true selectivity by over 10× for complex predicates (see § 3.2.2). Therefore, we simply rely on the query semantics to estimate the complexity of the predicates. Specifically, if the predicate contains more than 10 clauses, we use random sampling instead of clustering to select sample partitions.

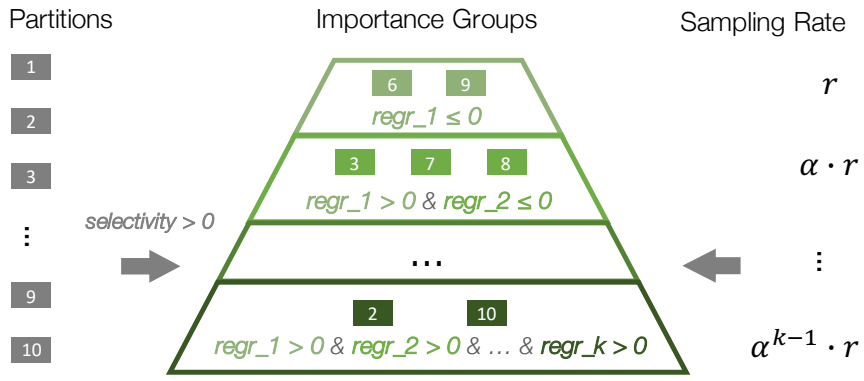


Figure 3.2: The trained regressors are used to classify input partitions into groups of different importance. The sampling rate decreases by a factor of $\alpha > 1$ from the i^{th} important to the $(i+1)^{th}$ important group.

3.3.3 Learned Importance-Style Sampling

While clustering helps select partitions that are dissimilar, it makes no distinction between partitions that contribute more to the query and partitions that contribute less. Ideally, we would want to sample the more important partitions more frequently to reduce the variance of the estimate [147].

The feature vectors can help assess partition contribution. Consider the query: `SELECT SUM(X), Y FROM table WHERE Z > 1 GROUP BY Y`. The subset of partitions that answer this query well should contain large values of `X`, many rows that satisfy the predicate and many distinct values of `Y`. Feature vectors are correlated with these desired properties: measure statistics (e.g. max, std) can help reveal large values of `X`, selectivity measures the fraction of the partition that is relevant to the query, and heavy hitter and distinct value statistics summarize the distribution of groups. However, it is challenging to manually quantify how much each feature matters for each query. In our example, it is unclear whether a partition with a high variance of `X` but few rows that match the predicate should be prioritized over a partition with low variance and many rows that match the predicate.

While it is not obvious how to manually design rules that relate feature vectors to partition contribution, a model may *learn* to do so from examples. An intuitive design is to use partition features as inputs and predict partition weights as outputs, which turns out to be a non-traditional regression problem. The goal of the regressor is to assign a weight vector to N partitions such that the weighted partition choice produces a small approximation error. Given a sampling budget of n partitions, there are exponentially many choices of subsets of partitions of size n and the optimal choice is discontinuous on n. In addition, the decision depends jointly on the *set* of partitions chosen; the weight assigned to one partition, for example, may depend on how many other partitions with nearly identical content are picked in the sample. Therefore, a simple, per partition regressor is unable to capture the combinatorial nature of the decision space. Existing solutions [40, 188] would

require significantly more resources and we pursue a lightweight alternative instead.

Given the challenges to directly use learned models to predict sampling probabilities, we propose a design that utilizes the models indirectly for sample size allocation; similar observations were made for using learned models to improve row-level sampling designs for count queries [321]. We consider classifying partitions based on their *relative importance* to the query answer into a few importance groups, and apply multiplicatively increasing sampling probability to the more important groups. We detail each of these steps next.

Partition Contribution. We consider the "contribution" of a partition to the answer of a query as its largest relative contribution to any group and any aggregate in the answer. Recall that we denote the aggregates for group $g \in G$ as $\mathbf{A_g} \in \mathbb{R}^d$, where d is the number of the aggregate functions, and the aggregates for group g on partition i are denoted as $\mathbf{A_{g,i}} \in \mathbb{R}^d$. Partition i's contribution is defined as: $\max_{g \in G} \max_{j=1}^{d} (\frac{\mathbf{A_{g,i}}[j]}{\mathbf{A_g}[j]})$. There are several alternative definitions of contribution, such as using the average instead of the max of the ratios, or using absolute values instead of the relatives. Among all variants, the max of the relatives is perhaps the most generous: it recognizes a partition's importance if it helps with *any* aggregates in *any* groups, and is not biased towards large groups or aggregates with large absolute values. We find that our simple definition above already leads to good empirical results.

Training. Given the partition contributions for all queries in the training data, we train a set of k models to distinguish the relative importance of partitions. When k is large, training the set of models is equivalent to solving the regression problem in which we are directly predicting partition contribution from the feature vector; when k is small, the training reduces to a simpler multiclass classification problem. The k models discretize partition contribution into $k+1$ bins, and we choose exponentially spaced bin boundaries: the number of partitions that satisfy the i^{th}model increase exponentially from the number of partitions that satisfy the $(i+1)^{th}$ model. In particular, the first model identifies all partitions that have non zero contribution to the query and the last model identifies partitions whose contribution is ranked in the top 1% of all partitions[4]. We use the `XGBoost` regressor as our base model, and provide additional details of the training in Appendix A.1.2.

Testing. During test time, we run partitions through a funnel that utilizes the set of trained models as filters and sort partitions into different importance groups (Figure 3.2). The advantage of building a funnel is that it requires partitions to pass more filters as they advance to the more important groups, which help limit the impact of inaccurate models. We list the procedure in Algorithm 2. We start from all partitions with non zero `selectivity_upper` feature; as discussed in § 3.2.2, this filter has perfect recall but varying precision depending on the complexity of the predicates. We run the partitions through the first trained model, and move the ones that pass the model to the next stage in the funnel. We repeat this process, each time taking the partitions at the end of the funnel,

[4]The small number of positive examples make it challenging to train an accurate model beyond 1%.

Algorithm 2 Group partitions by importance.

Input: partition features F

```
function ImportanceGroup(F, parts, regressors)
    groups.add(FilterByPredicate(F, parts))
    for regr ∈ regressors do
        to_examine ← groups[-1]
        to_pick ← p ∈ to_examine s.t. regr(F[p]) > 0
        groups[-1] ← to_examine.difference(to_pick)
        groups.add(to_pick)
    end for
    return groups
end function
```

running them through a more restrictive filter (model) and advance ones that pass the filter into the next stage until we run out of filters.

We then split the sampling budget such that more important groups get a greater proportion of the budget. We implement a sampling rate that decays by a factor of $\alpha > 1$ from the i^{th} important to the $(i+1)^{th}$ important group. We investigate the impact of the decay rate α in the sensitivity analysis in § 3.4.5. In general, increasing α improves the overall performance especially when the trained models are accurate, but the marginal benefit decreases as α becomes larger. If the trained models are completely random however, a larger α would increase the variance of the estimate. We have found that a decay rate of $\alpha = 2$ with $k = 4$ models works well across a range of datasets and layouts empirically. However, it is possible to fine-tune α for each dataset to further improve the performance and we leave the fine-tuning to future work.

3.3.4 Outliers

Finally, we observe that datasets often exhibit significant skew in practice (example in the chapter's introduction). Prior work in AQP has shown that augmenting random samples with a small number of samples with outlying values or from rare groups helps reduce error caused by the skewness [27, 67]. We recognize the importance of handling outliers and allocate a small portion of the sampling budget for outlying partitions.

We are especially interested in partitions that contain a rare distribution of groups for `GROUP BY` queries. These partitions are not representative of other partitions and should be excluded from clustering. To identify such partitions, we take advantage of the occurrence bitmap feature that tracks which heavy hitters are present in a partition. We put partitions with identical bitmap features for columns in the `GROUP BY` clause in the same group and consider a bitmap feature group outlying if its size is small both in absolute (< 10 partitions) and relative terms ($< 10\%$ the size of the largest group). For example, if there are 100 such bitmap feature groups and 10 partitions per group, we do not consider any group as outlying although the absolute size of each group is small.

We allocate up to 10% of the sampling budget to evaluate outliers. We have empirically found that increasing the outlier budget further does not significantly improve the performance using only the outliers we consider. Exploring alternative ways to identify outliers could be an interesting area for improvement for future works.

3.4 Evaluation

In this section, we evaluate the empirical performance of PS^3. Experiments show that:

1. PS^3 consistently outperforms alternatives on a variety of real-world datasets, delivering $2.7-70\times$ reduction of data read to achieve the same average relative error compared to uniform partition sampling, with storage overhead ranging from 12KB to 103KB per partition.
2. Every component of PS^3 and every type of features contribute meaningfully to the final performance.
3. PS^3 works across datasets, partitioning schemes, partition counts and generalizes to unseen queries.

3.4.1 Experimental Setup

In this subsection, we describe the experimental methodology, which includes the datasets, query generation, methods of comparison and error metrics.

Datasets

We evaluate using four real-world datasets summarized below. We include a detailed specification of the table schema in Appendix A of the extended report [174].

TPC-H*. Data is generated from a Zipfian distribution with a skewness of 1 and a scale factor of 1000 [256]. We denormalize all tables against the *lineitem* table. The denormalized table can support 16 out of 22 queries in the TPC-H benchmark (Q1,3,4,5,6,7,8, 9,10,12,14,15,17,18,19,21). We additionally include two derived columns `L_YEAR` and `O_YEAR` in the view in order to support group by clauses on these columns (Q7,8,9). The resulting table has 6B rows, with 14 numeric columns and 31 categorical columns. Data is sorted by column `L_SHIPDATE`.

TPC-DS*. *catalog_sales* table with a scale factor of 1 from TPC-DS, joined with dimension tables *item*, *date_dim*, *promotion* and *customer_demographics*, with 4.3M rows, 21 numeric columns and 20 categorical columns. Data is sorted by columns `year`, `month` and `day`.

Aria. Production service request log at Microsoft with 10M rows, 7 numeric columns and 4 categorical columns [1,124]. Data is sorted by categorical column `TenantId`.

KDD. KDD Cup'99 dataset on network intrusion detection with 4.8M rows, 27 numeric columns and 14 categorical columns [35]. Data is sorted by numeric column `count`.

By default we use a partition count of 1000, the smallest size from which partition elimination becomes interesting. The `TPC-H*` dataset (sf=1000) has 2844 partitions, with a partition size of about 2.5GB, consistent with the scale of the big-data workloads seen in practice. In the sensitivity analysis, we further investigate the effect of the partition count (§ 3.4.5), and data layouts on the performance (§ 3.4.5).

Query Set

To train PS^3, we construct a training set of 400 queries for each dataset by sampling at random the following aspects:

- between 0 and 8 columns as the group by columns
- between 0 and 5 predicate clauses; each of which picks a column, an operator and a constant at random
- between 1 and 3 aggregates over one or more columns

We generate a held-out set of 100 test queries in a similar way. For `TPC-H*`, we include an additional test set of 10 TPC-H queries (§ 3.4.5). We ensure that there are no identical queries between the training and test sets and that there is substantial entropy in our choice of predicates, aggregates and grouping columns.

Methods of Comparison

All methods except for uniform random sampling have access to feature vectors, and use the `selectivity_upper` feature to filter out partitions that do not satisfy the predicate before sampling. Recall that this filter has false positives but no false negatives. All methods have access to the same set of features. We report the average of 10 runs for methods that use random sampling.

Random Sampling. Partitions are sampled uniformly at random. Aggregates in the answer are scaled up by the sampling rate.

Random+Filter. Same as random sampling except that only partitions that pass the selectivity filter are sampled. This is only achievable with the use of summary statistics.

Learned Stratified Sampling (LSS). A baseline inspired by prior work on learned row-level stratified sampling [321]. We rank partitions by the model's prediction and perform stratification such that each strata covers partitions whose predictions fall into a consecutive range. We made three modifications to LSS to enable partition-level sampling:

- We move the training from online to offline, and use one trained model *per dataset and layout* instead of *per query*. LSS performs training inline for each query, using a fixed portion of the

Table 3.3: Strata sizes for the modified LSS algorithm selected via exhaustive search.

	Sampling Budget (% data read)								
	10	20	30	40	50	60	70	80	90
TPC-H*	15	50	100	250	260	580	430	50	730
TPC-DS*	55	120	85	130	160	250	395	170	10
Aria	75	80	55	150	260	70	80	130	190
KDD	90	160	295	230	360	430	220	410	820

sampling budget as the training data. Training on random row-level samples may invalidate I/O gains and already require a full scan over data. Instead, we train the model offline on training queries sampled from the workload and use the same trained model for all test queries.

- We change the model's inputs and labels. LSS operates on rows, while we use partition features as inputs. LSS only considers count queries, so the label is either 0 or 1. To support aggregates and group bys, we use the partition contribution defined in § 3.3.3 as labels.
- We use different stratification strategies. Prior work analyzes optimal choices of strata boundaries for proportional allocation of samples, in which the sample size allocated to each stratum is proportional to its size. The analysis does not extend to our setup, so we use equi-width strata instead. To set the number of strata, we exhaustively sweep the strata sizes and select one that minimizes average relative error on the training set. We report the selected strata sizes in Table 3.3.

PS^3. A prototype that matches the description given so far. Unless otherwise specified, default parameter values for PS^3 in all experiments are $k = 4, \alpha = 2$ and up to a 10% sampling budget dedicated to outliers.

Error Metric

Similar to prior work [5, 27, 177], we report multiple accuracy metrics. It is possible, for example, for a method to have a small absolute error but miss all small groups and small aggregate values. We therefore consider all three metrics below for a complete picture.

Missed Groups. Percentage of groups in the true answer that are missed by the estimate.

Average Relative Error. The average of the relative error for each aggregate in each group. For missed groups, the relative error is counted as 1.

Absolute Error over True. The average absolute error value of an aggregate across groups divide by the average true value of the aggregate across groups, averaged over multiple aggregates.

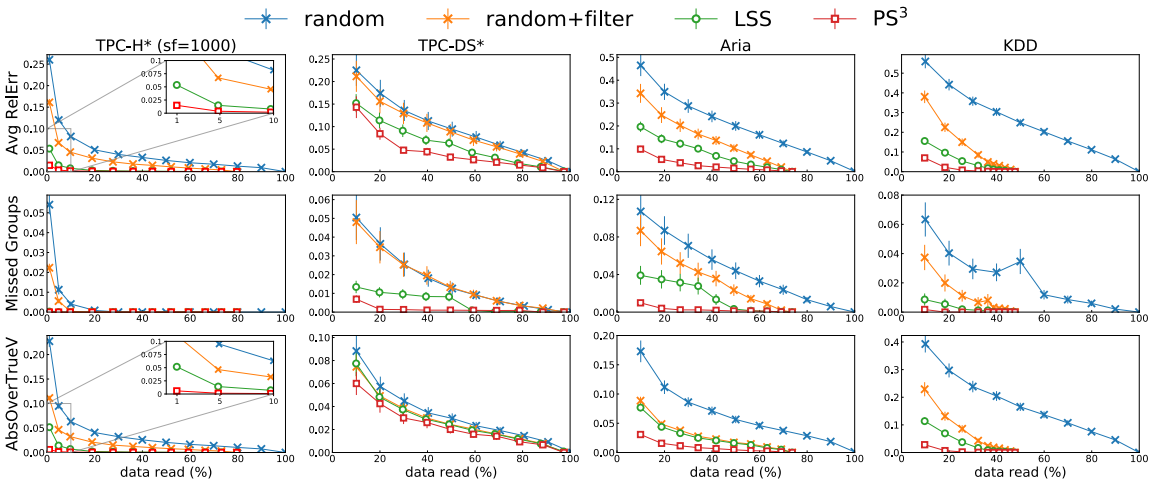


Figure 3.3: Comparison of error under varying sampling budget on four datasets, lower is better. PS^3 (red) consistently outperforms others across datasets and different error metrics.

3.4.2 Macro-benchmarks

We compare the performance of methods of interest under varying sampling budgets on four datasets (Figure 3.3). The closer the curve is to the bottom left, the better the results.

While the scale of the three error metrics is different, the ordering of the methods is relatively stable. Using the selectivity feature to filter out partitions that do not satisfy the predicate strictly improves the performance for all methods, except on datasets like `TPC-DS*` where most partitions pass the predicate. The modified LSS (green) improves upon random sampling by leveraging the correlation between feature vectors and partition contribution, consistent with findings of prior work.

Overall, PS^3 consistently outperforms alternatives across datasets and error metrics. On our large scale experiment with the `TPC-H*` data, PS^3 achieves an average relative error of 1.5% with a 1% sampling rate. With a 1% sampling rate, PS^3 improves the error achieved by 17.5× compared to random sampling, 10.8× compared to random sampling with filter and 3.6× compared to LSS (read from intersections between the baseline curves and a vertical line at 1% sampling rate). To achieve an average relative error of 1.5%, PS^3 reduces the fraction of data read by over 70× compared to random sampling, over 40× compared to random sampling with filter and 5× compared to LSS (read from the intersections between baseline curves and a horizontal line at 1.5% error rate). We observe similar trends on the three smaller datasets but the performance gap is smaller: PS^3 reduces the data read by 2.7× to 8.5× compared to random sampling to achieve $\leq 10\%$ average relative error.

We additionally show that the fraction of data read is a reliable proxy for reductions in resources used, measured by total compute time. We evaluate example queries on the `TPC-H*` dataset using SCOPE clusters [58, 343], Microsoft's main batch analytics platform, which consist of tens of thousands of nodes. Table 3.4 shows that reading 1%, 5% and 10% of the partitions results in a near

Table 3.4: Average speedups for query latency and total compute time under difference sampling rates on the TPC-H* dataset.

	1%	5%	10%	100%
Query Latency	4.7×	1.6×	1.5×	-
Total Compute Time	105.3×	19.6×	11.4×	-

Table 3.5: Per partition storage overhead of the summary statistics (in KB) for each dataset.

Dataset	Total	Histogram	HH	AKMV	Measure
TPC-H*	84.25	9.52	13.26	55.31	6.16
TPC-DS*	103.49	10.51	4.67	81.45	6.86
Aria	18.38	1.42	0.81	15.19	0.97
KDD	12.00	2.19	0.82	5.29	3.70

linear speedup of 105.3×, 19.6×, 11.4× in the total compute time. Improvement of query latency however, is less than linear and depends on stragglers and other concurrent jobs on the cluster.

3.4.3 Overheads

We report the space overhead of storing summary statistics in Table 3.5. The statistics are computed for each column and therefore require a constant storage overhead per partition. The overheads range from 12KB to 103KB across the four datasets. The larger the partition size, the lower the relative storage overhead of the statistics. For example, with a partition size of 2.5GB, the storage overhead is below 0.003% for the TPC-H* dataset.

The AKMV sketch for estimating distinct values takes the most space compared to other sketches. If the number of distinct values in a column is larger than k (we use $k = 128$), the sketch has a fixed size; otherwise the sketch size is proportional to the number of distinct values. The KDD dataset, for example, has more columns but a smaller AKMV sketch size compared to the Aria dataset since a number of its columns are binary.

We also report the single-thread latency of the partition picker (Algorithm 1) in Table 3.6, measured on an Intel Xeon E5-2690 v4 CPU. Our prototype picker is implemented in Python using the XGBoost and Sklearn libraries. Overall, the overhead is a small fraction of the query time,

Table 3.6: Range of the average picker overhead across sampling budgets for each dataset (in milliseconds).

	Aria	KDD	TPC-DS*	TPC-H*
Total	89.9±4.7	106.4±4.9	219.6±4.7	1002.1±13.3
Clustering	24.1±5.0	58.0±2.2	148.0±5.4	802.4±12.8

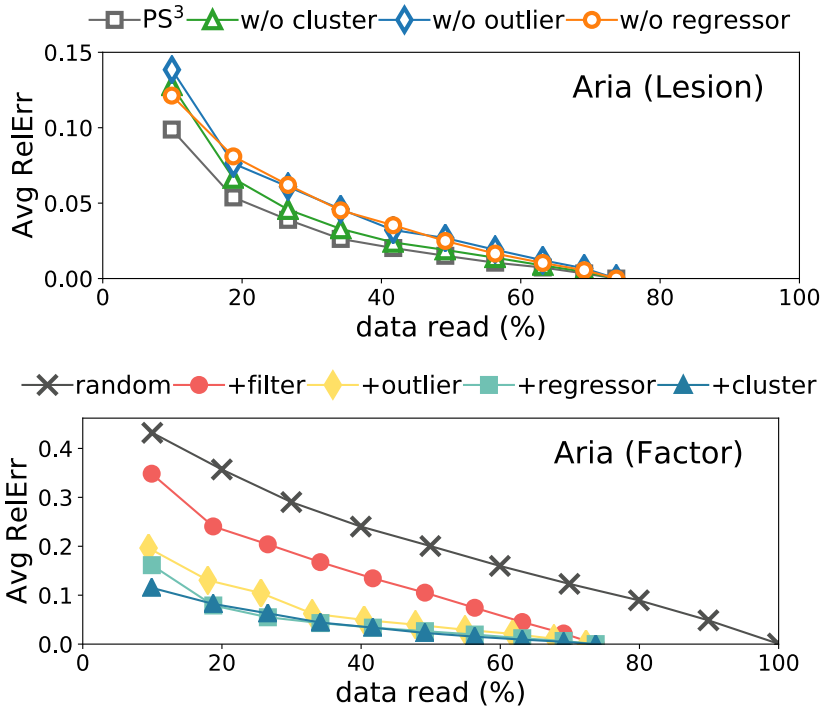


Figure 3.4: Lesion study and factor analysis on the Aria dataset. Each component of our system contributes meaningfully to the final accuracy. Results are similar on other datasets.

ranging from 86.5ms to around 1s across datasets. In comparison, the average query takes tens of total computation hours on the `TPC-H*` dataset. As the number of partitions and the dimension of the feature vectors increase, the total overhead increases and the clustering component takes up an increasing proportion of the overhead. The overhead can be further reduced via optimization such as performing clustering in parallel across different importance groups.

3.4.4 Lesion Study

In this section, we take a closer look at individual components of the picker and their impact on the final performance, as well as the importance of partition features.

Picker Lesion Study

We inspect how the three components of the partition picker introduced in § 3.3 impact the final accuracy. To examine the degree to which a single component impacts the performance, we perform a lesion study where we remove each component from the picker while keeping the others enabled (Figure 3.4, top). To disable clustering (§ 3.3.2), we use random sampling to select samples. To disable identification of outlier partitions (§ 3.3.4), we take away the sampling budget dedicated to

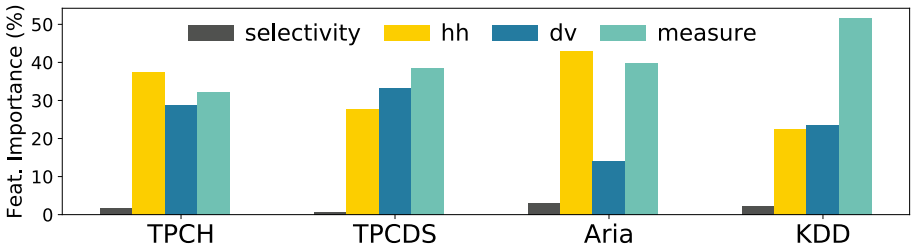


Figure 3.5: Feature importance for the regressors. The higher the percentage, the more important the statistics are to the regressor's accuracy.

outliers. To disable the regressor (§ 3.3.3), we apply the same sampling rate to all partitions. The result shows that the final error increases when each component is disabled, illustrating that each component is necessary to achieve the best performance.

We additionally measure how the three components contribute to overall performance. Figure 3.4 (bottom) reports a factor analysis. We start from the simple random sampling baseline (random). Using `selectivity_upper` ≥ 0 as a filter (+filter) strictly improves the performance. Similar to the lesion study, we enable each component on top of the filter (not cumulative) while keeping others disabled. The results show that the identification of outlier partitions (+outlier) contributes the least value individually and the use of clustering (+cluster) contributes the most.

Feature Importance

We divide partition features into four categories based on the sketches used to generate them: selectivity, heavy hitter, distinct value and measures. We investigate the contribution of features in each component of PS^3. The outlier component depends exclusively on the heavy hitter features. The clustering component uses all four feature types and we report the list of features selected for each dataset in Appendix A.1.1. For the learned component, we measure the regressors' feature importance via the "gain" metric, which reports the improvement in accuracy brought by a feature to the branches it is on [331]. For each dataset, we report the gain for features in each category as a percentage of the total gain aggregated over all learned models. The larger the percentage, the more important the feature is to the final accuracy. We report the result in Figure 3.5.

Overall, all four types of features contribute to the regressor accuracy, but the relative importance varies across the datasets. Selectivity estimates, despite being less useful for regressors, are useful to filter out partitions that do not contain any rows satisfying the predicate.

3.4.5 Sensitivity Analysis

In this section, we evaluate the sensitivity of the system's performance to changes in setups and key parameters.

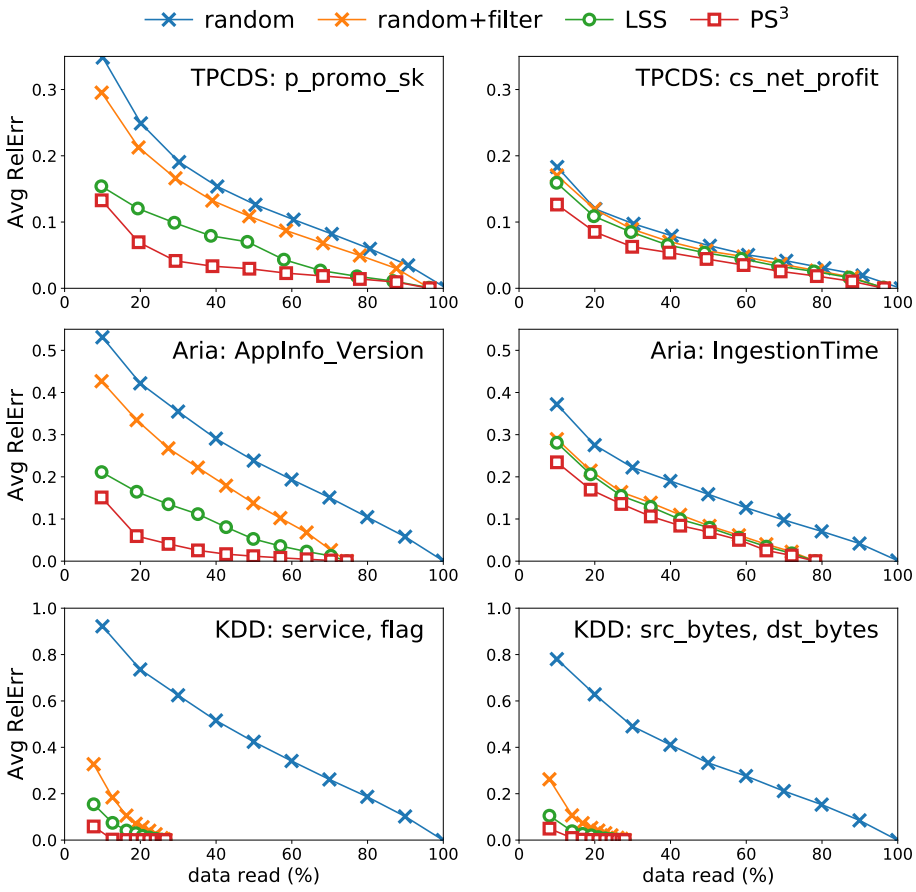


Figure 3.6: Our method consistently outperforms alternatives across datasets and data layouts.

Effect of Data Layouts

One of our design constraints is to be able to work with data in situ. To assess how PS^3 performs on different data layouts, we evaluate on two additional layouts for each dataset using the same training and testing query sets from experiments in § 3.4.2. Figure 3.6 summarizes the average relative error achieved under varying sampling budgets for the six combinations of datasets and data layouts.

PS^3 consistently outperforms alternatives across the board, but the sizes of the improvements vary across datasets and layouts. Overall, the more random the data layout is, the less room for improvement for importance-style sampling. In the `TPC-DS*` dataset for example, the layout sorted by column `cs_net_profit` is more uniform than the layout sorted by column `p_promo_sk`, since random sampling achieves a much smaller error under the same sampling budget in the former layout. LSS is only marginally better than random in the former layout, indicating a weak correlation between features and partition importance.

As a special case, we explicitly evaluate PS^3 on a random layout for the `TPC-H*` dataset with a

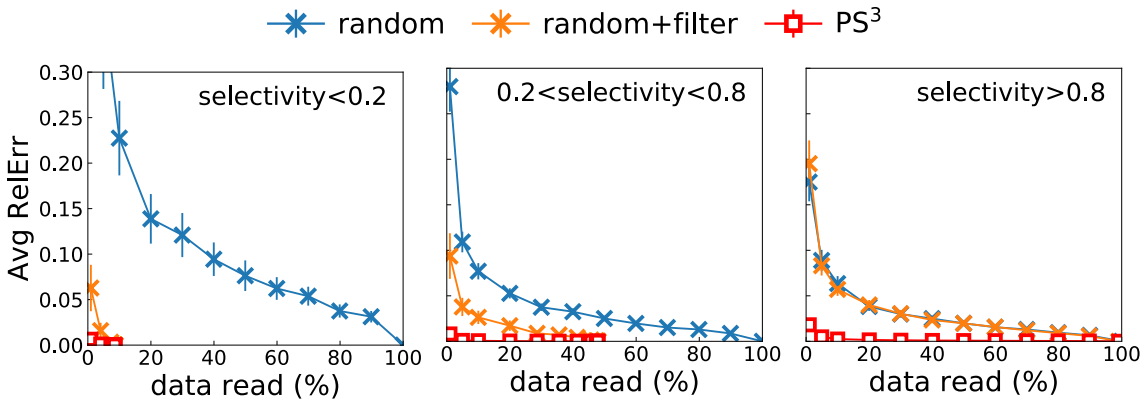


Figure 3.7: Performance breakdown by query selectivity on the `TPC-H*` dataset (sf=1000).

scale factor of 1 (Figure 3.8, left). As expected, sampling partitions uniformly at random performs well on the random layout. PS^3 underperforms random sampling in this setting, but the performance difference is small. Realistically, we do not expect PS^3 to be used for random data layouts; users would have chosen random sampling were they paying the cost to maintain a random data layout [54].

Effect of Query Selectivity

We investigate how queries with different sensitivities benefit from PS^3. Figure 3.7 reports the error breakdown by query selectivity for random partition-level sampling and PS^3 on the `TPC-H*` dataset; other datasets show similar trends. Compared to naive random partition level sampling (blue), PS^3 offers more improvements for more selective queries (selectivity < 0.2), since the selectivity feature effectively filters out a large fraction of partitions that are irrelevant to the query. Compared to random partition level sampling with the selectivity filter (orange), PS^3 offers more improvements for non-selective queries (selectivity > 0.8), since they have larger errors at small sampling rates.

Effect of Partition Count

In this subsection, we investigate the impact of partition count on the final performance. We report results on the `TPC-H*` dataset (sf=1) with 1000 and 10,000 partitions in the middle and right plot of Figure 3.8. Compared to results on the same dataset with fewer partitions, the percentage of partitions that can be skipped increases with the increase in the number of partitions. In addition, as the partition count increases, the error achieved under the same sampling fraction becomes smaller. However, the overheads of PS^3 also increase with the number of partitions. Specifically, the storage overhead for per-partition statistics increases linearly with the partition count and the latency of the partition picker also increases. Perhaps more concerning is the increase in I/O costs. The larger the partition count, the smaller the size of each partition. In the limit when each partition only

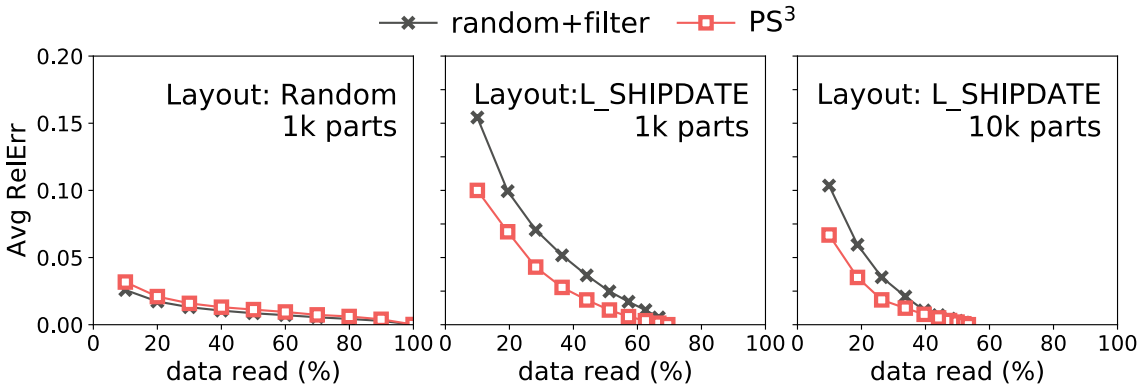


Figure 3.8: Comparison of `TPC-H*` (sf=1) results on different data layouts and total number of partitions.

contains one row, partition-level sampling is equivalent to row-level sampling, which is expensive to construct as discussed earlier.

Generalization Test on TPC-H Queries

To further assess the ability of the trained models to generalize to unseen queries, we test PS^3 trained on the randomly generated training queries with TPC-H schema (described in § 3.4.1) on 10 unseen TPC-H queries supported by our query scope[5]; the set of aggregate functions and group by columnsets are shared between the train and test set. To support Q8 and Q14, we rewrite the `SUM` aggregate with a `CASE` condition as an aggregate over the predicate. In addition, PS^3 explicitly chooses to use random sampling instead of clustering to select samples for Q19, which has complex predicates consisting of 21 clauses (§ 3.3.2). Our training queries are sampled randomly according to procedure described in § 3.4.1. We generate 20 random test queries for each TPC-H query template.

We report the detailed performance across test TPC-H queries on the `TPC-H*` dataset (sf=1000) in Figure 3.9. On average, PS^3 is still able to outperform uniform partition sampling, despite the larger domain gap between training and test set compared to experiments conducted in § 3.4.2. Overall, PS^3 significantly outperforms random partition selection on Q1, Q6 and Q7, and performs similarly to random partition selection on other queries. In particular, Q1, Q6 and Q7 all have a small number of partitions with either rare groups or outlying aggregate values. While PS^3 can identify such partitions via clustering and outlier detection, random partition selection can easily miss these important partitions especially when the sampling budget is limited. Our improvements are limited on Q8 which has a more complex aggregate and a nested query.

[5] Q4 is excluded since it operates on the *orders* table.

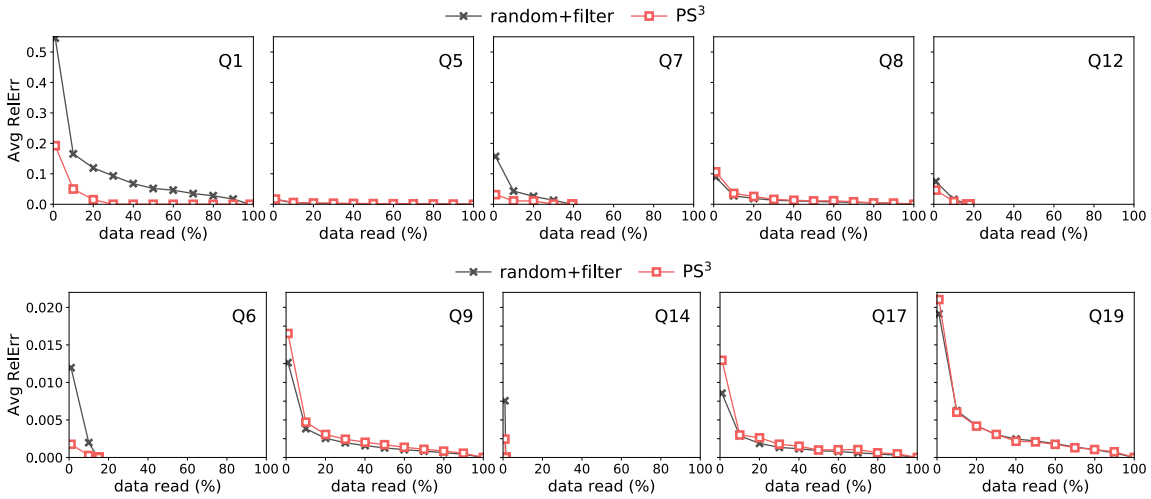


Figure 3.9: Detailed breakdown of results on TPC-H queries used in the generalization test. Overall, PS^3 significantly outperforms random partition selection on Q1, Q6, Q7 and performs similarly to random partition selection on other queries.

Table 3.7: AUC for different clustering algorithms; smaller is better.

	HAC(single)	HAC(ward)	KMeans
TPCDS	12.1	4.2	4.2
Aria	3.2	2.6	2.7
KDD	.71	.58	.55

Effect of Clustering Algorithm

We evaluate the effect of clustering algorithm choice on the clustering performance. We compare a bottom-up clustering algorithm (Hierarchical Agglomerative Clustering, or HAC) to a top-down algorithm (KMeans). For HAC, we evaluate two linkage metrics: the "single" linkage minimizes the minimum distances between all points of the two merged clusters, while the "ward" linkage minimizes the variances of two merged clusters. For each dataset, we evaluate the *average relative error* for estimating the query answer, and report the area under the error curve under different sampling budgets (Table 3.7). The smaller the area, the better the clustering performance. We also include results from a similar evaluation on the impact of the feature selection procedure in Appendix A.1.1.

HAC using the "ward" linkage metric and K-Means consistently produce similar results, suggesting that the clustering performance is not dependent on the choice of the clustering algorithm. The single linkage metric, however, produces worse results especially on the TPCDS dataset.

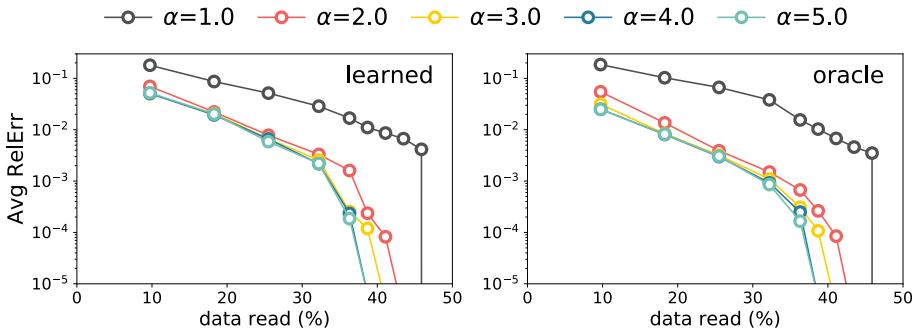


Figure 3.10: Impact of the sampling decay rate α on the KDD dataset. Larger α improves performance, but the marginal benefits decreases.

Effect of Sampling Rate

We investigate the extent to which applying different sampling rates affects the performance of learned importance style sampling. Recall that we tune the sampling rate via parameter α, which is the ratio of sampling rates between the i^{th} important and the $(i+1)^{th}$ important group. The larger α is, the more samples we allocate to the important groups. We report the results achieved under different αs for the `KDD` dataset (Figure 3.10, left). Overall the performance improves with the increase of α, but the marginal benefit decreases.

We repeat the experiment and replace the trained regressors with an oracle that has perfect precision and recall (Figure 3.10, right). This gives an upper bound of the improvements enabled by important-styled sampling. Compared to using learned models, the overall error decreases with the oracle, as expected. The performance gap between the learned and the oracle regressor increases with the increase of α. The comparison shows that the more accurate the regressor, the more benefits we get from using higher sampling rates for important groups. While we used a default value of $\alpha = 2$ across the experiments, it is possible to fine-tune α for each dataset to improve performances.

3.5 Discussion

3.5.1 Related Work

We discuss related work in sampling-based AQP, data skipping, and partition-level sampling.

Learning to Sample. Prior works have used learning to improve the sampling efficiency for AQP. One line of work uses learning to model the dataset and reduce the number of samples required to answer queries [98]. Along similar lines, properties of the dataset distribution can also be learned through the answers to past queries, which can then be used to refine answers to new queries [242]. Our work is closer to works that use learned models to improve the design of the sampling scheme.

Recent work proposes a learned stratified sampling scheme wherein the model predictions are used as stratification criteria [321]. However, the work focuses on row-level samples and on count queries; we support a broader scope of queries with aggregates and group bys and work with partition-level samples. In the evaluation, we compare against a scheme inspired by learned stratified sampling.

Data Skipping. Our work is also closely related to prior works on data skipping which studied the problems of optimizing data layouts [305, 307, 335, 339] and indexing [165, 166, 280], improving data skipping given a query workload. Building on the observation that it is often difficult, if not impossible, to find a data layout that offers optimal data skipping for all queries, we instead choose to work with data in situ. Researchers and practitioners have also looked at ways to use statistics and metadata to prune partitions that are irrelevant to the query. The proposed approaches range from using simple statistics such as min and max to check for predicate ranges [167, 230], to deriving complex pruning rules for queries with joins [178]. Our work is directly inspired by this line of work and extends deterministic pruning to probabilistic partition selection.

Partition-level sampling. Researchers have long recognized the I/O benefits of partition-level sampling over row-level sampling [159, 283]. Partition-level samples have been used to build statistics such as histograms and distinct value estimates for query optimizers [69, 72]. Prior work has studied combining row-level and partition-level Bernoulli style sampling for `SUM, COUNT`, and `AVG` queries, in which one can adjust the overall sampling rate but each sample is treated equally [143]. Our work more closely resembles importance sampling where we sample more important partitions with higher probability.

Partition level sampling is also studied in the context of online aggregation (OLA) where query estimates can be progressively refined as more data gets processed, and users can stop the processing when the answer reaches target accuracy [79, 241, 287]. Classic work in OLA assumes that tuples are processed in a random order, which often require random shuffling as an expensive processing step [54]. Our approach does not require random layout, and in fact, should not be used if the data layout is random. Prior work has also studied OLA over raw data, which requires an expensive tuple extraction step to process raw files [78]. PS^3 can work with data stored in any format as long as per-partition statistics are available and focuses on selecting fewer partitions instead of stopping processing early within a partition, since the most expensive operation for our setup is the I/O cost of reading the partition.

3.5.2 Future Directions

Our work shows promise as a first step towards using summary statistics to improve upon uniform partition-level sampling. We highlight a few important areas for future work.

First, our system is designed mainly for read-only and append-only data stores, so the proposed set of sketches should be reconsidered if deletions and edits to data must be supported. Furthermore,

the partition picker logic must be retrained when the summary statistics of partitions change in a substantial way.

Second, our work only considers generalization to unseen queries in the same workload on the same dataset and data layout. Although retraining can help generalize to unseen columns in the same dataset and layout, supporting broader forms of generalization such as to different data layouts is non-trivial and requires further attention.

Third, our work demonstrates empirical advantages to uniform partition-level sampling on several real-world datasets but provides no apriori error guarantees. Enabling users to work with an error budget in addition to a sampling budget and developing diagnostic procedures for failure cases will be of immediate value to practitioners.

3.6 Conclusion

In this chapter, we introduced PS^3, a system that novelly combines precomputed summary statistics with weighted query-time partition selection in big-data clusters. We propose a set of sketches – measures, heavy hitters, distinct values, and histograms – to generate partition-level summary statistics that help assess partition similarity and importance. We show that our prototype PS^3 provides sizable speed ups compared to random partition selection with a small storage overhead.

Chapter 4

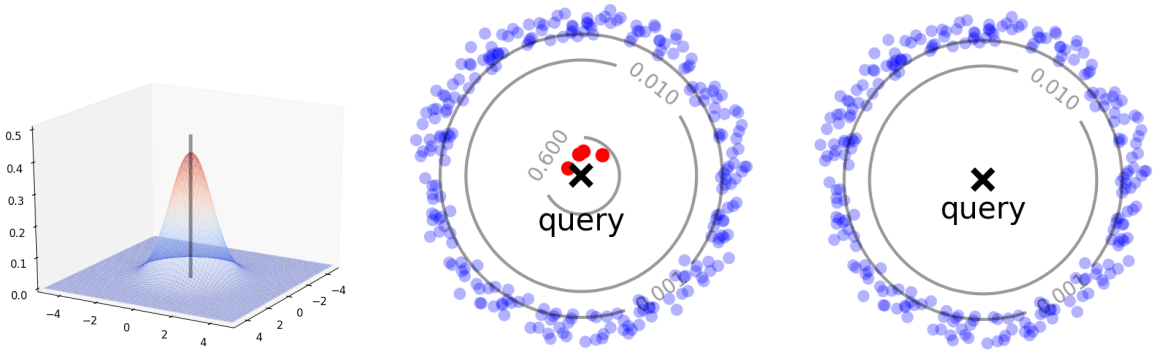
HBE: Approximate Kernel Density Estimation with Hashing

This chapter presents a practical system that approximates Kernel Density Estimation (KDE) using precomputed hash indexes as a smart data sampler. KDE is a non-parametric method to estimate the probability density function of a random variable. Given a dataset $P = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$, a kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$, and a vector of (non-negative) weights $u \in \mathbb{R}^n$, the weighted KDE at $q \in \mathbb{R}^d$ is given by $\text{KDE}_P^u(q) := \sum_{i=1}^n u_i k(q, x_i)$. KDE has been used in a wide range of applications, such as clustering [155,312], classification [162,228], anomaly detection [100,196], and domain science [153,271,315]. For example, KDE was one of the outlier detection methods used in our anomaly detection and explanation system MacroBase [29]. However, the evaluation time of KDE increases quadratically with the dataset size, which limits its applicability to large-scale high-dimensional datasets. This work aims to, after some precomputation, efficiently estimate the KDE at a query point with $(1 \pm \epsilon)$ multiplicative accuracy under a small failure probability δ.

The approximate KDE problem is well-studied [137,138,219], but in high dimensions, only recently novel importance sampling algorithms, referred to as *Hashing-Based-Estimators (HBEs)*, have demonstrated provable improvements over random sampling (RS) under worst-case assumptions [63]. At its core, HBE uses a hash function h (randomized space partition) to construct for any q an unbiased estimator of $\mu := \text{KDE}_P^u(q)$.[1] [63] showed that given a hash function with evaluation time T and collision probability $\mathbb{P}[h(x) = h(y)] = \Theta(\sqrt{k(x, y)})$, it is possible to obtain a $(1 \pm \epsilon)$ approximation to $\mu \geq \tau$ in time $\tilde{O}(T/\epsilon^2\sqrt{\mu})$ and space $\tilde{O}(n/\epsilon^2\sqrt{\tau})$, thus improving over random sampling, which requires time $O(1/\epsilon^2\mu)$ in the worst case.

HBE improves upon RS by better sampling points with larger weights (kernel values). In particular, RS does not perform well (Figure 4.1b) when a significant portion of the density comes from

[1] The value $\mu \in (0, 1]$ can be seen as a *margin* for the decision surface $\sum_{i=1}^n u_i k(q, x_i) \geq 0$.



(a) kernel (b) difficult case (c) simple case

Figure 4.1: (a) For radially decreasing kernels (e.g. Gaussian), points close to the query have higher kernel values than those that are far. (b) Random sampling does not perform well when a small number of close points contribute significantly to the query density. (c) Random sampling performs well when most points have similar kernel values (distance from query).

a few points with large weights (close to the query). HBE samples uniformly from a set of biased samples (hash buckets where the query is mapped to) that has a higher probability of containing high-weight points. For radially decreasing kernels (Figure 4.1a), the biased sample can be produced via LSH [168]. To obtain m such biased samples for estimating the query density, the scheme requires building m independent hash tables for the entire dataset. The runtime mentioned above is achieved by setting $m = O(1/\epsilon^2\sqrt{\mu})$. In practice, one cares about values of $\mu \geq 1/\sqrt{n}$, which is a lower bound on the order of statistical fluctuations when P consists of n i.i.d samples from some smooth distribution [303].

Despite progress on the worst-case query time, the idea of using hashing for kernel evaluation, introduced independently in [300] and [63], has largely been confined to the theoretical realm due to a few practical concerns. First, straightforward implementations of the proposed approach require a large amount (e.g., $\tilde{O}(n^{\frac{5}{4}})$ for $\tau = 1/\sqrt{n}$) of precomputation time and memory to create the requisite number of hash tables. Second, RS can outperform HBE on certain datasets (Figure 4.1c), a fact that is not captured by the worst-case theoretical bounds. Third, to implement this approach [63] uses an adaptive procedure to estimate the sufficient sample size m. For this procedure to work, the estimators must satisfy some uniform polynomial decay of the variance as a function of μ. Only the hashing scheme of [17] (with an significant $\exp(O(\log^{\frac{2}{3}} n))$ runtime and memory overhead per hash table) has been shown to satisfy this criteria for the popular Gaussian kernel. These issues call the applicability of HBE into question and to the best of our knowledge, the method has not been previously shown to empirically improve upon competing methods.

In this work, we demonstrate that the hashing index approach is relevant for real-world datasets. We bridge the gap between theory and practice by making the following contributions (Figure 4.2):

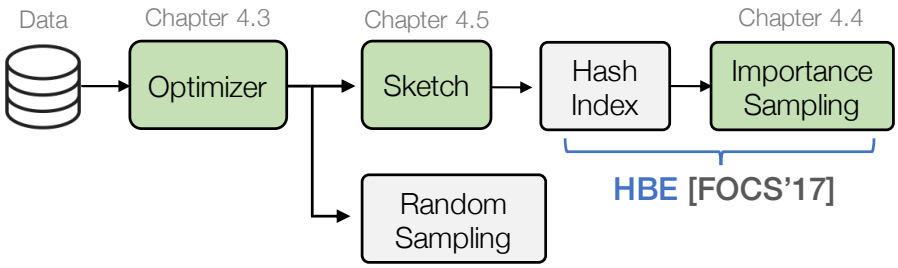


Figure 4.2: To make HBE practical, we improved the constant factor of its sampling algorithm (Section 4.3), developed a sketch to reduce its precomputation overhead (Section 4.4), and introduced an optimizer to switch to random sampling when HBE underperforms (Section 4.2).

1. **Precomputation overhead reduction via sketching.** HBE requires superlinear precomputation time and memory as a result of having to hash the entire dataset once for each sample. To reduce this overhead, we develop new theoretical results on sketching the KDE by a weighted subset of points using hashing and non-uniform sampling (Section 4.4). Our hashing-based sketch (HBS) can better sample sparse regions of space while still maintaining a variance close to that of a random sketch, leading to better performance on low-density points. Applying HBE on top of the sketch results in considerable gains without sacrificing accuracy.

2. **Cost-based optimizer.** Despite worst-case guarantees, in practice RS outperforms HBE on certain datasets . To quantify this phenomenon, we introduce an inequality that bounds the variance of unbiased estimators including HBE and RS (Section 4.2). We propose a query optimizer inspired procedure utilizing the variance bounds that, for a given dataset, choose at runtime with minimal overhead the better of the two approaches and do so without invoking HBE; our evaluation showed that the optimizer is both accurate and efficient. The new variance bound also allowed us to simplify and recover the theoretical results of [63].

3. **A practical sampling algorithm for the Gaussian kernel.** The adaptive sampling procedure of [63] estimates the sufficient sample size m in parallel with estimating the density μ. We design a simplified version of this procedure and provide an improved analysis that results in an order of magnitude speedup in the query time (Section 4.3). More importantly, our new analysis demonstrates that slightly weaker conditions (non-uniform but bounded polynomial decay) on the variance of the estimators are sufficient for the procedure to work. By proving that HBE based on the hashing scheme of [93] satisfies the condition, we propose the first practical algorithm that provably improves upon RS for the Gaussian kernel in high dimensions. It was not known previously how to use this hashing scheme within an adaptive procedure for this purpose.

We perform extensive experimental evaluations of our methods on a variety of synthetic benchmarks as well as real-world datasets. Our evaluations against other state-of-the-art competing

methods for kernel evaluation show that:

- HBE outperforms all competing methods for synthetic "worst-case" instances with multi-scale structure and dimensions $d \geq 10$, as well as for "structured" instances with a moderate number of clusters (Section 4.5.1).
- For real-world datasets, HBE is always competitive with alternative methods and up to $\times 10$ faster for many datasets compared to the next best method (Section 4.5.2).

In summary, our theoretical and experimental results constitute an important step toward making HBE practical and in turn improving the scalability of kernel evaluations in large high-dimensional data sets.

The remainder of this chapter is organized as follows. Section 4.1 presents basic definitions and prior work which we build upon. Section 4.2 describes new variance bounds to support the cost-based optimizer. Section 4.3 provides an improved adaptive sampling algorithm that is instrumental in obtaining a practical HBE for the Gaussian kernel, and Section 4.4 demonstrates how to reduce the precomputation overheads through sketching. Section 4.5 presents the experimental evaluation and Section 4.6 concludes with discussions of related work and future directions.

4.1 Preliminaries

In this section, we discuss related work, present the basic definitions and give a self-contained presentation of Hashing-Based-Estimators that summarizes the parts of [63] upon which we build. All material beyond Section 4.1 is new.

Notation. For a set $S \subset [n]$ and numbers $u_1, \ldots, u_n$, let $u_S := \sum_{i \in S} u_i$. Let $\Delta_n := \{u \in \mathbb{R}^n_+ : \|u\|_1 = 1\}$ denote the n-dimensional simplex. Throughout the paper we assume that we want to approximate KDE^u_P with $u \in \Delta_n$. We can handle the general case $u \in \mathbb{R}^d$ by treating the positive and negative part of u separately.

Kernels. All our theoretical results, unless explicitly stated, apply to general non-negative kernels. For concreteness and in our experiments, we focus on the Gaussian $\exp(-\|x-y\|^2/\sigma^2)$ and Laplace kernels $\exp(-\|x-y\|/\sigma)$ [38], and typically suppress the dependence on the bandwidth $\sigma > 0$ (equivalently, we can rescale our points).

4.1.1 Multiplicative Approximation & Relative Variance

Definition 1. *A random variable $\hat{Z}$ is called an (ϵ, δ)-approximation to μ if $\mathbb{P}[|\hat{Z} - \mu| \geq \epsilon\mu] \leq \delta$.*

Given access to an unbiased estimator $\mathbb{E}[Z] = \mu$, our goal is to output an (ϵ, δ)-approximation to μ. Towards that end the main quantity to control is the *relative variance*.

Definition 2. *For a non-negative random variable Z we define the* relative variance *as* $\text{RelVar}[Z] := \frac{\text{Var}[Z]}{\mathbb{E}[Z]^2} \leq \frac{\mathbb{E}[Z^2]}{\mathbb{E}[Z]^2}$.

The relative variance captures the fluctuations of the random variable at the scale of the expectation. This is made precise in the following lemma that combines Chebyshev's and Paley-Zygmund inequality.

Lemma 1. *For a non-negative random variable Z and parameters $t > 0$, $\theta \in [0, 1]$, we have:*

$$\mathbb{P}[Z \geq (t+1) \cdot \mathbb{E}[Z]] \leq \frac{1}{t^2} \cdot \text{RelVar}[Z], \tag{4.1}$$

$$\mathbb{P}[Z > (1-\theta)\mathbb{E}[Z]] \geq \frac{1}{1 + \frac{1}{\theta^2} \cdot \text{RelVar}[Z]}. \tag{4.2}$$

As $\text{RelVar}[Z]$ decreases, the upper bound in (4.1) decreases while the lower bound in (4.2) increases, increasing our overall confidence of Z being an accurate estimate of the mean. Thus, if one can construct a random variable Z with $\mathbb{E}[Z] = \mu$ and small relative variance $\text{RelVar}[Z] = O(\epsilon^2)$, Lemma 1 shows that Z is an $(\epsilon, O(1))$-approximation to μ. In fact, by Chernoff bounds, one can use the median-trick to boost the probability of success (Appendix B.1).

4.1.2 Hashing-Based-Estimators (HBEs)

HBE uses hashing to create a data-structure that, after some precomputation, can produce unbiased estimators for the KDE at query time (Figure 4.3) with low variance. Let $\mathcal{H}$ be a set of functions and ν a distribution over $\mathcal{H}$. We denote by $h \sim \mathcal{H}_\nu$ a random function $h \in \mathcal{H}$ sampled from ν and refer to $\mathcal{H}_\nu$ as a hashing scheme.

Definition 3. *Given a hashing scheme $\mathcal{H}_\nu$, we define the collision probability between two elements $x, y \in \mathbb{R}^d$ as $p(x, y) := \mathbb{P}_{h \sim \mathcal{H}_\nu}[h(x) = h(y)]$.*

Precomputation: given dataset P and hashing scheme $\mathcal{H}_\nu$:

- Sample m hash functions $h_t \overset{i.i.d.}{\sim} \mathcal{H}_\nu$ for $t \in [m]$.
- Create hash tables $H_t := h_t(P)$ for $t \in [m]$ by evaluating the hash functions on P.

The m hash tables allow us to produce at most m independent samples to estimate the KDE for each query.

Query-time Sampling: query $q \in \mathbb{R}^d$, hash table index $t \in [m]$

- let $H_t(q) := \{i \in [n] : h_t(x_i) = h_t(q)\}$ denote the hash bucket (potentially empty) that q maps to.
- If $H_t(q)$ is not empty return $Z_{h_t} := \frac{k(X_t, q)}{p(X_t, q)} u_{H_t(q)}$ where X_t is sampled with probability proportional to u_i from $H_t(q)$, otherwise return 0.

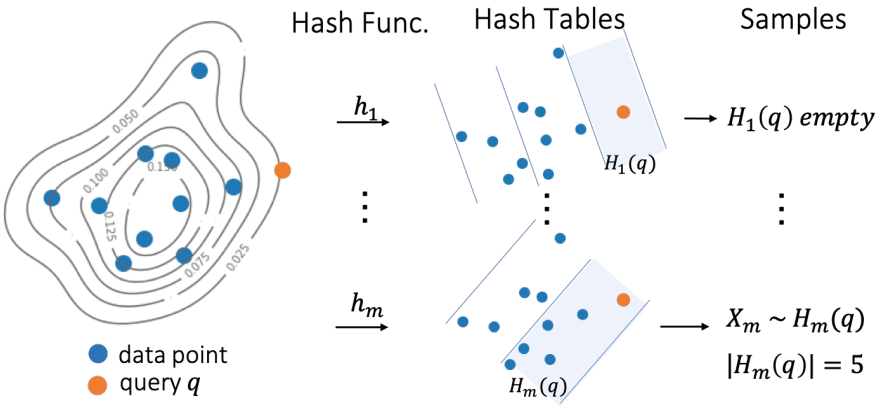


Figure 4.3: Given a dataset, the HBE approach samples a number of hash functions and populates a separate hash table for each hash function. At query time, for each hash table, we sample a point at random from the hash bucket that the query maps to.

By averaging many samples produced by the hash tables we get accurate estimates. The salient properties of the estimators Z_{h_t} are captured in the following lemma.

Lemma 2. *Assuming that* $\forall i \in [n], p(x_i, q) > 0$ *then*

$$\mathbb{E}[Z_h] = \sum_{i=1}^{n} u_i k(x, x_i), \tag{4.3}$$

$$\mathbb{E}[Z_h^2] = \sum_{i,j=1}^{n} k^2(q, x_i) \frac{u_i \mathbb{P}[i, j \in H(q)] u_j}{p^2(q, x_i)}. \tag{4.4}$$

As we see in (4.4), the second moment depends on the hashing scheme through the ratio $\frac{\mathbb{P}[i,j \in H(q)]}{p^2(q,x_i)}$. Thus, we hope to reduce the variance by selecting the hashing scheme appropriately. The difficulty lies in that the variance depends on the positions of points in the whole dataset. [63] introduced a property of HBE that allowed them to obtain provable bounds on the variance.

Definition 4. *Given a kernel* k*, a hashing scheme is called* (β, M)*-scale-free for* $\beta \in [0, 1]$ *and* $M \geq 1$ *if:* $\frac{1}{M} \cdot k(x, y)^\beta \leq p(x, y) \leq M \cdot k(x, y)^\beta$.

Thus, one needs to design a hashing scheme that "adapts" to the kernel function. Next, we present a specific family of hashing schemes that can be used for kernel evaluation under the Exponential and Gaussian kernel.

4.1.3 HBE via Euclidean LSH

In the context of solving Approximate Nearest Neighbor search in Euclidean space, Datar et al. [93] introduced the following hashing scheme, *Euclidean LSH* (eLSH), that uses two parameters $w > 0$

(width) and $\kappa \geq 1$ (power). First, hash functions in the form of $h_i(x) := \lceil \frac{g_i^\top x}{w} + b_i \rceil$ map a d dimensional vector x into a set of integers ("buckets") by applying a random projection ($g_i \overset{i.i.d.}{\sim} \mathcal{N}(0, I_d)$), a random shift ($b_i \overset{i.i.d.}{\sim} U[0,1]$)) and quantizing the result (width $w > 0$). A concatenation (power) of κ such functions gives the final hash function.

They showed that the collision probability for two points $x, y \in \mathbb{R}^d$ at distance $\|x - y\| = c \cdot w$ equals $p_1^\kappa(c)$ where

$$p_1(c) := 1 - 2\Phi(c^{-1}) - \sqrt{\frac{2}{\pi}} c \left(1 - \exp\left\{-\frac{c^{-2}}{2}\right\}\right). \tag{4.5}$$

In order to evaluate the hashing scheme on a dataset P, one needs space and pre-processing time $O(d\kappa \cdot n)$. By picking w large enough, i.e. for small c, one can show the following bounds on the collision probability.

Lemma 3 ([63]). *For $\delta \leq \frac{1}{2}$ and $c \leq \min\{\delta, \frac{1}{\sqrt{\log(\frac{1}{\delta})}}\}$: $e^{-\sqrt{\frac{2}{\pi}}\delta} \leq \frac{p_1(c)}{e^{-\sqrt{\frac{2}{\pi}}c}} \leq e^{\sqrt{\frac{2}{\pi}}\delta^3}$.*

Taking appropriate powers κ, one can then use Euclidean LSH to create collision probabilities appropriate for the Exponential and Gaussian kernel.

Theorem 1 ([63]). *Let $\mathcal{H}_1(w, \kappa)$ denote the eLSH hashing scheme with width w and power $\kappa \geq 1$. Let $R = \max_{x,y \in P \cup \{q\}} \{\|x - y\|\}$. Then for the*

- ***Laplace kernel,*** *setting $\kappa_e = \lceil \sqrt{2\pi} \log(\frac{1}{\tau}) R \rceil$ and $w_e = \sqrt{\frac{2}{\pi}} \frac{1}{\beta} \kappa_e$ results in $(\beta, \sqrt{e})$-scale free hashing scheme for $k(x, y) = e^{-\|x-y\|_2}$.*
- ***Gaussian kernel,*** *setting $r_t := \frac{1}{2}\sqrt{t \log(1 + \gamma)}$, $\kappa_t := 3\lceil r_t R \rceil^2$, $w_t = \sqrt{\frac{2}{\pi}} \frac{\kappa_t}{r_t}$ results in an estimator for $k(x, y) = e^{-\|x-y\|^2}$ with relative variance*

$$\mathrm{RelVar}[Z_t] \leq V_t(\mu) := 4e^{\frac{3}{2}} \frac{1}{\mu} e^{r_t^2 - r_t \sqrt{\log(\frac{1}{\mu})}}. \tag{4.6}$$

4.2 Cost-based Optimizer

HBE's theoretical guarantees are only for worst-case scenarios, so it is not surprising that RS can outperform HBE on certain datasets in practice. In this section, we develop new bounds on the relative variances of RS and HBE to quantify the data-dependent performance differences. We then introduce a query optimizer inspired procedure utilizing the bounds that, for a given dataset, chooses at runtime with minimal overhead the better of the two approaches.

4.2.1 Refined Bounds on Relative Variance

To understand how many samples are needed to estimate the density through RS or HBE, we need bounds for expressions of the type (4.4). In particular, for a sequence of numbers $w_1, \ldots, w_n \in [0, 1]$,

e.g. $w_i = k(q, x_i)$, and $u \in \Delta_n$ such that $\mu = \sum_{i\in[n]} u_i w_i$, we want to bound:

$$\sup_{u\in\Delta_n, u^\top w=\mu} \sum_{i,j\in[n]} w_i^2 \left(u_i V_{ij} u_j\right) \tag{4.7}$$

where $V \in \mathbb{R}_+^{n\times n}$ is a non-negative matrix. Our bound will be decomposed into the contribution μ_ℓ from a subset of indices $S_\ell \subseteq [n]$ where the weights (kernel values) have a bounded range. Specifically, let $\mu \le \lambda \le L \le 1$ and define: $S_1 = \{i \in [n] : L \le w_i \le 1\}, S_2 = \{i \in [n] \setminus S_1 : \lambda \le w_i \le L\}, S_3 = \{i \in [n] \setminus (S_2 \cup S_1) : \mu \le w_i \le \lambda\}, S_4 = \{i \in [n] : w_i < \mu\}$ as well as $\mu_\ell = \sum_{i\in S_\ell} u_i w_i \le \mu$ for $\ell \in [4]$. The intuition behind the definition of the sets is that for radial decreasing kernels, they correspond to spherical annuli around the query.

Lemma 4. *For non-negative weights $w_1, \dots, w_n$, vector $u \in \Delta_n$ and sets $S_1, \dots, S_4 \subseteq [n]$ as above it holds*

$$\begin{aligned}\sum_{i,j\in[n]} w_i^2\{u_i V_{ij} u_j\} \le &\sum_{\ell\in[3],\ell'\in[3]} \sup_{\substack{i\in S_\ell,\\ j\in S_{\ell'}}} \left\{\frac{V_{ij} w_i}{w_j}\right\} \mu_\ell \mu_{\ell'} \\ &+ u_{S_4} \sum_{\ell\in[3]} \sup_{\substack{i\in S_\ell,\\ j\in S_4}} \left\{V_{ij}\frac{w_i}{\mu}\right\} \mu_\ell \mu \\ &+ \sup_{i\in S_4, j\in[n]} \{V_{ij} w_i\} \cdot \mu_4 \end{aligned} \tag{4.8}$$

where $u_S := \sum_{j\in S} u_j \le 1$.

This simple lemma allows us to get strong bounds for scale-free hashing schemes, simplifying results of [63] and extending them to $\beta < 1/2$.

Theorem 2. *Let $\mathcal{H}_\nu$ be (β, M)-scale free hashing scheme and Z_h the corresponding HBE. Then* $\mathrm{RelVar}[Z_h] \le V_{\beta,M}(\mu) := 3M^3/\mu^{\max\{\beta,1-\beta\}}$.

Proof. Letting $w_i = k(q, x_i)$, we start from (4.4) and use the fact that $\mathbb{P}[i, j \in H(q)] \le \min\{p(x_i, q), p(x_j, q)\}$ and that the hashing scheme is (β, M)-scale free, to arrive at $\mathbb{E}[Z_h^2] \le \sum_{i,j\in[n]} w_i^2\{u_i V_{ij} u_j\}$ with $V_{ij} = M^3 \frac{\min\{w_i^\beta, w_j^\beta\}}{w_i^{2\beta}}$. Let $S_1, \dots, S_4$ be sets defined for $\ell = L = \mu$. Then $S_2 = S_3 = \emptyset$. By Lemma

4 we get

$$\begin{aligned}\mathbb{E}[Z_h^2] &\leq M^3\Big(\sup_{i,j\in S_1}\{\frac{w_i^{1-2\beta}}{w_j^{1-\beta}}\}\mu_1^2 + \sup_{i\in S_4}\{w_i^{1-\beta}\}\mu_4 \\ &\qquad + u_{S_4}\sup_{i\in S_1, j\in S_4}\{w_i^{1-2\beta}w_j^{\beta}\}\mu_1\Big) \\ &\leq M^3\Big(\frac{1}{\mu^{1-\beta}}\mu_1^2 + u_{S_4}\sup_{i\in S_1}\{\frac{w_i^{1-2\beta}}{\mu^{1-\beta}}\}\mu\mu_1 + \mu^{1-\beta}\mu_4\Big) \\ &\leq M^3\Big(\frac{1}{\mu^{1-\beta}} + \frac{1}{\mu^{\max\{\beta,1-\beta\}}} + \frac{1}{\mu^{\beta}}\Big)\mu^2.\end{aligned}$$

Noting that $\mathbb{E}[Z_h] = \mu$ and $\mu \leq 1$ concludes the proof. □

Remark 1. *It is easy to see that random sampling is a (trivial)* $(0,1)$*-scale free hashing scheme, hence the relative variance is bounded by* $\frac{3}{\mu}$.

Remark 2. *For any* $(\frac{1}{2}, M)$*-scale free hashing scheme the relative variance is bounded by* $\frac{3M^3}{\sqrt{\mu}}$.

4.2.2 Comparing Dataset Dependent Performance

The inequality provides means to quantify the dataset dependent performance of RS and HBE: by setting the coefficients V_{ij} appropriately, Lemma 4 bounds the variance of RS ($V_{ij} = 1$) and HBE ($V_{ij} = \frac{\min\{p(q,x_i),p(q,x_j)\}}{p(q,x_i)^2}$). However, evaluating the bound over the whole dataset is no cheaper than evaluating the methods directly.

Instead, we evaluate the upper bound on a "representative sample" $\tilde{S}_0$ in place of $[n]$. By doing this for a number T of random queries picked uniformly from the dataset, we get an estimate of the average relative variance for different methods. Specifically, given $\tau \in (0,1)$ and $\epsilon \in (0,1)$, for a single query let $\tilde{S}_0$ be the random set produced by the AMR procedure (Algorithm 5) called with random sampling and define the sets $\tilde{S}_\ell = \tilde{S}_0 \cap S_\ell$ for $\ell \in [4]$ and their corresponding "densities" $\tilde{\mu}_\ell = \sum_{i\in\tilde{S}_\ell} u_i w_i$. Let

$$\lambda_\epsilon := \arg\max_{\tilde{\mu}_0 \leq \lambda \leq 1}\left\{\tilde{\mu}_3 \leq \frac{1}{2}(\epsilon\tilde{\mu}_0 - \tilde{\mu}_4)\right\}, \tag{4.9}$$

$$L_\epsilon := \arg\min_{\tilde{\mu}_0 \leq L \leq 1}\left\{\tilde{\mu}_1 \leq \frac{1}{2}(\epsilon\tilde{\mu}_0 - \tilde{\mu}_4)\right\}. \tag{4.10}$$

be such that $\tilde{\mu}_2 \geq (1-\epsilon)\tilde{\mu}_0$, i.e. most of the mass is captured by the set $\tilde{S}_2$ (that is an spherical annulus for kernels that are decreasing with distance). Since Lemma 4 holds for all $\mu \leq \lambda \leq L \leq 1$, $L_\epsilon, \lambda_\epsilon$ complete the definition of four sets on $\tilde{S}_0$, which we use to evaluate the upper bound. Finally, we produce a representative sample $\tilde{S}_0$ by running our adaptive procedure (Section 4.3) with Random Sampling. The procedure returns a (random) set $\tilde{S}_0$ such that $\tilde{\mu}_0$ is an $(\epsilon, O(1))$-approximation to μ for any given query.

Algorithm 3 Cost-based Optimizer

Input: set P, threshold $\tau \in (0,1)$, accuracy ϵ, $T \geq 1$, collision probability $p(x,y)$ of the hashing scheme $\mathcal{H}_\nu$.
for $t = 1, \ldots, T$ **do**
  $q \leftarrow \text{Random}(P)$ ▷ For each random query
  $(\tilde{S}_0, \tilde{\mu}_0) \leftarrow \text{AMR}(\mathcal{Z}_{RS}(q), \epsilon, \tau)$ ▷ Algorithm 5
  Set $\lambda_\epsilon, L_\epsilon$ using (4.9) and (4.10)
  $V_{\text{RS}} \leftarrow$ r.h.s of (4.8) for $\tilde{S}_0$ and $V_{ij} = 1$.
  $V_{\text{H}} \leftarrow$ r.h.s of (4.8) for $\tilde{S}_0$, $V_{ij} = \frac{\min\{p(q,x_i), p(q,x_j)\}}{p(q,x_i)^2}$.
  $\text{rV}_{\text{RS}}(t) \leftarrow V_{\text{RS}} / \max\{\tilde{\mu}_0, \tau\}^2$
  $\text{rV}_{\text{HBE}}(t) \leftarrow V_{\text{H}} / \max\{\tilde{\mu}_0, \tau\}^2$.
end for
Output: $(\text{mean}_T(rV_{\text{RS}}), \text{mean}_T(rV_{\text{HBE}}))$

Remark 3. *We only show the procedure for choosing between RS and HBE with a specific hashing scheme. The same procedure can be used to evaluate a multitude of hashing schemes to select the best one for a given dataset.*

In addition, we can use the information from our diagnostics to visualize the dataset by aggregating local information for random queries. For a set S, let $r_S = \min_{i \in S} \log(\frac{1}{k(x_i,q)})$ and $R_S = \max_{j \in S} \log(\frac{1}{k(x_j,q)})$. The basis of our visualization is the following fact:

Lemma 5. *Let X be a random sample from S, then $E[k^2(X,q)] \leq \exp(R_S - r_S) \cdot \mu_S^2$.*

Proof of Lemma 5. We have that $e^{-R_S} \leq k(x_i,q) \leq e^{-r_S}$, therefore

$$\mathbb{E}[k^2(X,q)] = \sum_{i \in S} k^2(x_i,q) u_i \leq e^{-r_S} \sum_{i \in S} k(x_i,q) u_i \frac{\mu_S}{\mu_S} \leq e^{R_S - r_S} \mu_S^2 \tag{4.11}$$

where in the last part we used $\mu_S \geq e^{-R_S}$. □

Thus if we plot an annulus of width $w_S = R_S - r_s$ then e^{w_S} is an estimate of the relative variance for RS. The visualization procedure when given a sequence of T pairs of numbers (λ_t, L_t) for $t \in [T]$ (produced by the optimizer procedure) plots overlapping annuli around the origin representing the queries. Since often the ratio $\max_{i,j \in S} \frac{k(x_i,q)}{k(x_j,q)}$ is referred to as the *condition number* of the set S, we call our procedure the Log-Condition plot.

Remark 4. *In the specific case of the Laplace (exponential) kernel, the radii we are plotting correspond to actual distances.*

4.3 Query-time: Adaptive Sampling

Towards our goal of producing an $(\epsilon, O(1))$-approximation to μ for a given query q, the arguments in Section 4.1.1 reduce this task to constructing an unbiased estimator Z of low-relative variance

Algorithm 4 Log-Condition Plot

Input: $\{(\lambda_t, L_t)\}_{t\in[T]}$.
for $t = 1, \ldots, T$ **do** ▷ For each query
  $r_t \leftarrow \log(1/L_t)$,
  $R_t \leftarrow \log(1/\lambda_t)$
  draw 2D-annulus(r_t, R_t)
end for
Output: figure with overlapping annuli.

$\text{RelVar}[Z] \leq \frac{\epsilon^2}{6}$. Given a basic estimator Z_0 and bound $V(\mu)$ on its relative variance, one can always create such an estimator by taking the mean of $O(V(\mu)/\epsilon^2)$ independent samples. The question that remains is *how to deal with the fact that μ and thus $V(\mu)$ is unknown?*

We handle this with an improved version of the adaptive sampling procedure of [63]. The procedure makes a guess of the density and uses the guess to set the number of samples. Subsequently, the procedure performs a consistency check to evaluate whether the guess was correct up to a small constant and if not, revises the guess and increases the sample size. The procedure applies *generically* to the following class of estimators (whose variance varies polynomially with μ) that we introduce.

4.3.1 (α, β, γ)-regular Estimators

Definition 5. *For $\alpha, \beta \in (0, 2]$ and $\gamma \geq 1$, an estimator $\mathcal{Z}$ is (α, β, γ)-regular if for some constant $C \geq 1$ and all $t \in [T]$, there exist a random variable Z_t and a function $V_t : (0, 1] \to \mathbb{R}_{++}$ such that*

(A) $\mathbb{E}[Z_t] = \mu$ *and* $\text{RelVar}[Z_t] \leq V_t(\mu)$, $\forall \mu \in (0, 1]$,

(B) $1 \leq \frac{V_t(y)}{V_t(x)} \leq \left(\frac{x}{y}\right)^{2-\alpha}$ *for all* $x \geq y > 0$,

(C) $V_t(\mu_{t+1}) \leq C\mu_t^{-\beta}$ *with* $\mu_t := (1 + \gamma)^{-t}$.

We write $Z_t \sim \mathcal{Z}(t, \gamma)$ to denote a sample from such an estimator at level $t \in [T]$.

At a high level, regular estimators generalize properties of scale-free estimators relevant to adaptively estimating the mean. Property (B) affects the success probability whereas (C) affects the running time and space requirements. This level of generality will be necessary to analyze the hashing scheme designed for the Gaussian kernel.

Regular Estimators via HBE. For any $t \in [T]$, given a collection of hash tables $\{H_t^{(i)}\}_{i\in[m_t]}$ created by evaluating m_t i.i.d hash functions with collision probability $p_t(x, y)$ on a set P and a fixed kernel $k(x, y)$, we will denote by

$$\mathcal{Z}(t, \gamma) \leftarrow \text{HBE}_{k,p_t}(\{H_t^{(i)}\}_{i\in[m_t]}) \tag{4.12}$$

a data structure at level t, that for any query q is able to produce up to m_t i.i.d unbiased random variables $Z_t^{(i)}(q)$ for the density μ according to Section 4.1.2. The union of such data structures for

$t \in [T]$ will be denoted by $\mathcal{Z}$.

Before proceeding with the description of the estimation procedure we show that (β, M)-scale free HBEs (that include random sampling) satisfy the above definition.

Theorem 3. *Given a (β, M)-scale free hashing scheme with $\beta \geq \frac{1}{2}$, the corresponding estimator is $(2-\beta, \beta, \gamma)$-regular with constant $C = 3M^3(1+\gamma)^\beta$.*

The proof follows easily by Theorem 2 by using the same estimator for all $t \in [T]$ and $V_t(\mu) = V(\mu) = \frac{3M^3}{\mu^\beta}$.

Algorithm 5 Adaptive Mean Relaxation (AMR)

Input: (a, β, γ)-regular estimator $\mathcal{Z}$, accuracy $\epsilon \in (0,1)$, threshold $\tau \in (0,1)$.
$T \leftarrow \lceil \log_{1+\gamma}(\frac{1}{\epsilon\tau}) \rceil + 1$
for $t = 1, \ldots, T$ **do** ▷ For each level
  $\mu_t \leftarrow (1+\gamma)^{-t}$ ▷ Current guess of mean
  $m_t \leftarrow \lceil \frac{6}{\epsilon^2} V_t(\mu_{t+1}) \rceil$ ▷ sufficient samples in level t
  $Z_t^{(i)} \sim \mathcal{Z}(t, \gamma)$ i.i.d samples for $i \in [m_t]$.
  $\bar{Z}_t \leftarrow \text{mean}\{Z_t^{(1)}, \ldots, Z_t^{(m_t)}\}$
  if $\bar{Z}_t \geq \mu_t$ **then** ▷ consistency check
    return $\bar{Z}_t$
  end if
end for
return 0 ▷ In this case $\mu \leq \epsilon\tau$

4.3.2 Adaptive Mean Relaxation

For regular estimators we propose the following procedure (Algorithm 5). In Appendix B.1 we analyze the probability that this procedure successfully estimates the mean as well as the number of samples it uses and obtain the following result.

Theorem 4. *Given an (a, β, γ)-regular estimator $\mathcal{Z}$, the AMR procedure outputs a number $\hat{Z}$ such that*

$$\mathbb{P}[|\hat{Z} - \mu| \leq \epsilon \cdot \max\{\mu, \tau\}] \geq \frac{2}{3} - O_{\gamma,\alpha}(\epsilon^2)$$

and with the same probability uses $O_\gamma(\frac{1}{\epsilon^2}\frac{1}{\mu^\beta})$ samples.

Using the bounds on the relative variance in Section 4.2, we get that any $(\beta \geq 1/2, M)$-scale free estimator can be used to estimate the density μ using $O(1/\epsilon^2\mu^\beta)$ samples.

Next, we show a construction of a HBE for Gaussian kernel (Algorithm 6) that results in a *regular estimator*.

Theorem 5. *$\mathcal{Z}_{\text{Gauss}}$ is $(1, \frac{3}{4}, \gamma)$-regular and takes preprocessing time/space bounded by $O_{d,\kappa_T,\gamma}(\epsilon^{-3+\frac{1}{4}}\tau^{-\frac{3}{4}} \cdot n)$.*

Algorithm 6 Gaussian HBE (Gauss-HBE)

Input: $\gamma \geq 1$, $\tau \in (0,1)$, $\epsilon \in (0,1)$, dataset P.
$T \leftarrow \lceil \log_{1+\gamma}(\frac{1}{\epsilon\tau}) \rceil + 1$, $R \leftarrow \sqrt{\log(1/\epsilon\tau)}$
for $t = 1, \ldots, T$ **do**
  $m_t \leftarrow \lceil \frac{6}{\epsilon^2} V_t((1+\gamma)^{-(t+1)}) \rceil$ ▷ see (4.6)
  for $i = 1, \ldots, m_t$ **do**
    $H_t^{(i)} \leftarrow \mathrm{eLSH}(w_t, \kappa_t, P)$ ▷ see Theorem 3
    $p_t(x,y) := p_1^{\kappa_t}(\|x-y\|/w_t)$ ▷ see (4.5)
    $k(x,y) := e^{-\|x-y\|^2}$
  end for
  $\mathcal{Z}_{\mathrm{Gauss}}(t,\gamma) \leftarrow \mathrm{HBE}_{k,p_t}(\{H_t^{(i)}\}_{i\in[m_t]})$
end for
Output: $\mathcal{Z}_{\mathrm{Gauss}}$

The proof (Appendix B.1) is based on using (4.6) to show that Definition 5 is satisfied with appropriate selection of constants. We also note that (4.6) can be derived using Lemma 4.

4.4 Precomputation: Reducing Overheads via Sketching

A different bottleneck in evaluating the KDE on large datasets is the space usage. For uniform random sampling the space usage is $O_d(n)$, whereas for the HBE approach is $O_{\epsilon,d}(\tau^{-O(1)}n)$, which can be prohibitively large for small values of τ. In this section, we introduce an approach based on hashing to create a "sketch" of KDE that we can evaluate using HBE or other methods. The motivation behind sketching is that we can use a large enough (weighted) subsample S of the dataset P and only introduce a small increase in the variance of our estimators applied to the subsample.

Definition 6. *Let (u,P) be a set of weights and points. We call (w,S) an (ϵ,δ,τ)-sketch iff for any $q \in \mathbb{R}^d$:*

$$\mathbb{E}[|\mathrm{KDE}_S^w(q) - \mathrm{KDE}_P^u(q)|^2] \leq \epsilon^2\tau\delta \cdot \mathrm{KDE}_P^u(q). \tag{4.13}$$

Let $\mu = \mathrm{KDE}_P^u(q)$, using Chebyshev's inequality it is immediate that any (ϵ,δ,τ)-sketch satisfies for any $q \in \mathbb{R}^d$: $\mathbb{P}[|\mathrm{KDE}_S^w(q) - \mu| \geq \epsilon \max\{\tau,\mu\}] \leq \delta$.

Remark 5. *It is easy to see that one can construct such a sketch by random sampling $m \geq \frac{1}{\epsilon^2\delta}\frac{1}{\tau}$ points.*

Hashing-Based-Sketch (HBS). The scheme we propose samples a random point by first sampling a hash bucket H_i with probability $\propto u_{H_i}^{\gamma}$ and then sampling a point j from the bucket with probability $\propto u_j$. The weights are chosen such that the sample is an unbiased estimate of the density. This scheme interpolates between uniform over buckets and uniform over the whole data set as we vary $\gamma \in [0,1]$. We pick γ^* so that the variance is controlled and with the additional property that any non-trivial bucket $u_{H_i} \geq \tau$ will have a representative in the sketch with some probability. This

last property is useful in estimating low-density points (e.g. for outlier detection [123]). For any hash table H and a vector $u \in \Delta_n$ (simplex), let $B = B(H)$ denote the number of buckets and $u_{\max} = u_{\max}(H) := \max\{u_{H_i} : i \in [B]\}$ the maximum weight of any hash bucket of H. The precise definition of our Hashing-Based-Sketch is given in Algorithm 7.

Theorem 6. *Let H be the hash function sampled by the HBS procedure. For $\epsilon > 0$ and $\delta \in [e^{-\frac{6}{\epsilon^2}\frac{u_{\max}}{n\tau}}, e^{-\frac{6}{\epsilon^2}})$, let:*

$$\gamma^* = \left\{1 - \frac{\log(\frac{\epsilon^2}{6}\log(1/\delta))}{\log(\frac{u_{\max}}{\tau})}\right\}^{\mathbb{I}[B \le (\frac{1}{2})^{\frac{1}{6}}\frac{1}{\tau}]}, \tag{4.14}$$

$$m = \frac{6}{\epsilon^2}\frac{1}{\tau}(Bu_{\max})^{1-\gamma^*} < \frac{\log(\frac{1}{\delta})}{\tau}. \tag{4.15}$$

Then (S_m, w) is an $(\epsilon, \frac{1}{6}, \tau)$-sketch and if $B \le (\frac{1}{2})^{\frac{1}{6}}\frac{1}{\tau}$ any hash bucket with weight at least τ will have non empty intersection with S_m with probability at least $1 - \delta$.

Algorithm 7 Hashing-Based-Sketch (HBS)

Input: set P, sketch size m, hashing scheme $\mathcal{H}_\nu$, threshold $\tau \in (0,1)$, $u \in \Delta_n$
Sample $h \sim \mathcal{H}_\nu$ and create hash table $H = h(P)$.
Set γ according to (4.14)
$S_m \leftarrow \emptyset$, $w \leftarrow 0 \cdot \mathbf{1}_m$, $B \leftarrow B(H)$
for $j = 1, \ldots, m$ **do**
    Sample hash bucket H_i with probability $\propto u_{H_i}^\gamma$
    Sample a point X_j from H_i with probability $\propto u_j$
    $S_m \leftarrow S_m \cup \{X_j\}$
    $w_j(\gamma, m) \leftarrow \frac{u_{H_i}}{m}\frac{\sum_{i'=1}^{B} u_{H_{i'}}^\gamma}{u_{H_i}^\gamma}$
end for
Output: (S_m, w)

4.5 Evaluation

In this section, we evaluate the performance of hashing-based methods on kernel evaluation on real and synthetic datasets, the optimizer's ability to predict dataset-dependent estimator performance, and the quality of the sketching procedure[2].

4.5.1 Experimental Setup

Baselines and Evaluation Metrics. As baselines for density estimation (using $u = \frac{1}{n}\mathbf{1}$), we compare against ASKIT [220], a tree-based method FigTree [226], and random sampling (RS). We

[2]Source code available at: `http://github.com/kexinrong/rehashing`

used the open-source libraries for FigTree and ASKIT; for comparison, all implementations are in C++ and we report results on a single core. We tune each set of parameters via binary search to guarantee an average relative error of at most 0.1. By default, the reported query time and relative error are averaged over 10K random queries that we evaluate exactly as ground truth. The majority of query densities in evaluation are larger than $\tau = 10^{-4}$ (roughly $\frac{1}{10\sqrt{n}}$).

Synthetic Benchmarks. To evaluate algorithms under generic scenarios, we need benchmarks with diverse structures such that they are simultaneously hard for different methods. The variance bound suggests that having a large number of points far from the query is hard for HBE, whereas having a few points close to the query is hard for RS. In addition, the performance of space-partitioning methods depends mostly on how clustered vs spread-out the datasets are, with the latter being harder. A fair benchmark should therefore include all above regimes.

We propose a procedure that generates a d-dimensional dataset where for D random directions, clusters of points are placed on s different distance scales, such that *i)* each distance scale contributes equally to the kernel density at the origin, *ii)* the outermost scale has n points per direction, *iii)* the kernel density around the origin is approximately μ. By picking $D \gg n$ the instances become more random like, while for $D \ll n$ they become more clustered. Using this procedure, we create two families of instances:

- **"worst-case":** we take the union of two datasets generated for $D = 10, n = 50\text{K}$ and $D = 5\text{K}$, $n = 100$ while keeping $s = 4$, $\mu = 10^{-3}$ fixed and varying d.
- **D-structured:** we set $N = 500\text{K}$, $s = 4$, $\mu = 10^{-3}$, $d = 100$ and vary D while keeping $nD = N$.

In fact, the "worst-case" instances are hard for all methods, since accurate estimation of the density at the origin requires handling both highly "clustered" points ($D \ll n$) and "scattered" points ($D \gg n$) at multiple scales ($s > 1$). In Appendix B.3 we include a precise description of the procedure as well as example scatter plots.

4.5.2 Performance on Real and Synthetic Datasets

Synthetic datasets. We evaluate the four algorithms for kernel density estimation on worst-case instances with $d \in [10, 500]$ and on D-structured instances with $D \in [1, 100K]$. In the experiments, the dataset sizes range from $0.5M$ to $1M$ and query points have density close to 10^{-3}.

We first investigate the impact of dimension on performances of these methods. We generated five synthetic datasets using the $(\mu, k, n, s, d, \sigma)$-Instance (Section 4.5.1) with fixed origin density ($\mu = 10^{-3}$), total number of data points ($N = 1M$), and varying dimensions ranging from 10 to 500. While all methods experience increased query time with increased dimensions, HBE achieves the best query time overall and is up to an order of magnitude faster than RS.

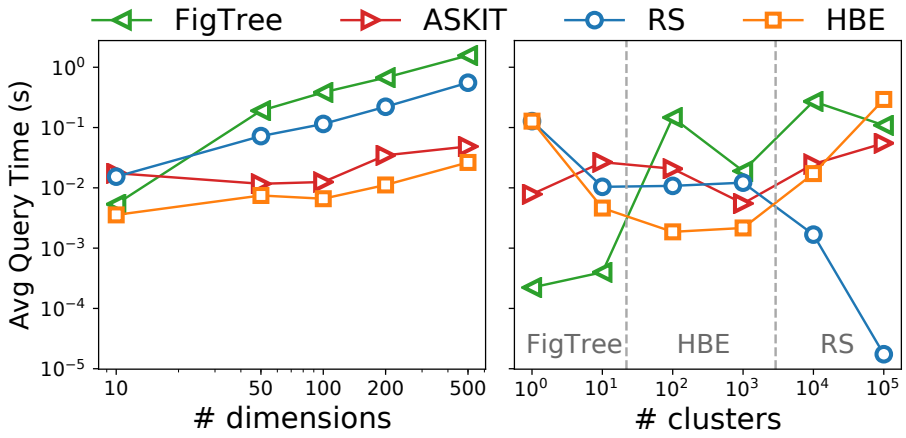


Figure 4.4: Evaluation of the impact of dimensions and cluster structures on query performances using synthetic benchmarks.

We further investigate the impact of cluster structures on performances. Similarly, we generated six synthetic datasets with a fixed dimension ($d = 100$), density ($\mu = 10^{-3}$), data points ($N = 500K$), and varying number of clusters (k) ranging from 1 to 10^5. Figure 4.4 (right) reports the average query time. FigTree dominates the region where the dataset is highly clustered while in contrast, random sampling performs the best when the dataset is highly scattered. For datasets with "intermediate" cluster structures, HBE has the best runtime. ASKIT's runtime is relatively unaffected by the number of clusters, but is outperformed by other methods in all settings.

Real-world datasets. We repeat the above experiments on eight large real-world datasets from various domains. We z-normalize each dataset dimension, and tune bandwidth based on Scott's rule [281]. We exclude a small percent of queries whose density is below τ. Table 4.1 reports the preprocessing time as well as average query time for each method. Overall, HBE achieves the best average query time on four datasets; we focus on query time since it dominates the runtime given enough queries. With sketching, HBE also exhibits comparable preprocessing overhead to FigTree and ASKIT. While the performances of FigTree and ASKIT degrade with the increased dimension and dataset size respectively, HBE remains competitive across the board.

We provide intuition on the performance differences between HBE and RS on these datasets using the visualization procedure (Figure 4.5). The main factor that affects the performance of random sampling is whether a significant portion of the density of the query comes from a relatively small number of points (e.g. census). HBE, on the other hand, is affected adversely when the contribution comes from a large number of points that are relatively far from the query (e.g. MSD).

Table 4.1: Comparison of preprocess time and average query time on real world datasets. Bold numbers correspond to the best result. The sources of the datasets are: MSD [43], GloVe [249], SVHN [234], TMY3 [154], covtype [46], TIMIT [126].

			Preprocess Time (sec)				Average Query Time (ms)			
Dataset	n	d	HBE	FigTree	ASKIT	RS	HBE	FigTree	ASKIT	RS
TMY3	$1.8M$	8	15	28	838	0	**0.8**	3.0	28.4	55.0
census	$2.5M$	68	66	4841	1456	0	**2.8**	1039.9	103.7	27.5
covertype	$581K$	54	22	31	132	0	**1.9**	23.0	47.0	149.4
TIMIT	$1M$	440	298	1579	439	0	**24.3**	1531.8	169.8	32.8
ALOI	$108K$	128	24	70	14	0	6.3	53.5	**5.4**	21.2
SVHN	$630K$	3072	2184	$> 10^5$	877	0	67.9	$> 10^4$	**43.0**	370.1
MSD	$463K$	90	55	2609	107	0	5.2	326.9	8.1	**2.1**
GloVe	$400K$	100	98	5603	76	0	19.0	656.5	86.1	**5.0**

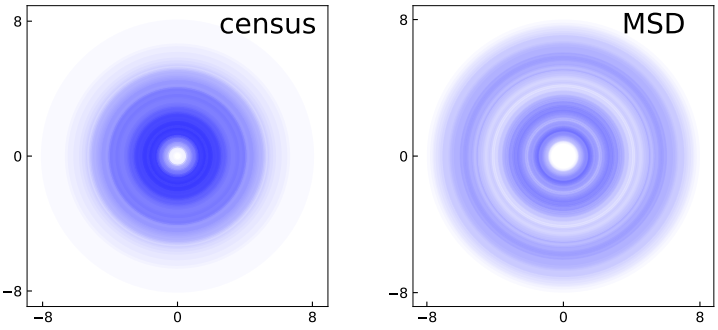


Figure 4.5: Illustration of the performance differences between HBE and RS.

4.5.3 Evaluation of the Cost-based Optimizer

To assess the accuracy of the optimizer, we compare the predicted variance of RS and HBE on an extended set of datasets. RS exhibits lower variances (smaller error with the same number of samples) for datasets on the left. Overall, our proposed optimizer correctly identified the better estimator for all but one dataset. Notice that although RS has better sampling efficiency on TIMIT, HBE ends up having better query time since the frequent cache misses induced by the large dataset dimension offset the additional computation cost of HBE. For all datasets in Table 4.1, the optimizer costs between 8% to 59% of the setup time of HBE, indicating that running the optimizer is cheaper than running even a single query with the HBE.

However, sampling efficiency does not necessarily translate directly to query performance. On moderate datasets, HBE queries can be a constant time factor slower (up to 8× in our experiments) than a RS query. Therefore, on datasets where RS and HBE have similar predicted variances (e.g. susy, poker, elevator), RS usually achieves better accuracy under the same *wallclock* time. On

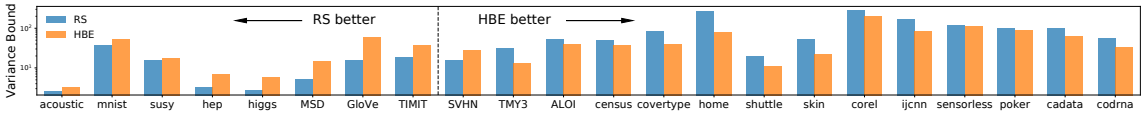


Figure 4.6: Predicted relative variance upper bound on real datasets. RS exhibits lower variance for datasets on the left. Overall, the optimizer procedure correctly predicts the performance of all but one dataset (SVHN).

datasets with large dimensions (e.g. TIMIT), frequent cache misses induced by random sampling offset the additional computation cost of HBE, so the latter ends up having better query time.

4.5.4 Evaluation of the Hashing-based Sketch

In this section, we evaluate the quality of the proposed hashing-based sketch. As baselines, we compare against sparse kernel approximation (SKA) [86], kernel herding algorithm (Herding) [77] and uniform sampling. To control for the difference in the complexity (Table 4.2), we compare the approximation error achieved by sketches of the same size (s) under the same compute budget ($2n$, where n is dataset size). We describe the detailed setup below, including necessary modifications to meet the computational constraints.

HBS. For HBS, we used 5 hash tables, each hashing a subset of $\frac{2}{5}n$ points in the dataset. In practice, we found that varying this small constant on the number of hash tables does not have a noticeable impact on the performance.

SKA. SKA (Algorithm 8) produces the sketch by greedily finding s points in the dataset that minimizes the maximum distance. The associated weights are given by solving an equation that involves the kernel matrix of the selected points. SKA's complexity $O(ns + s^3)$ is dominated by the matrix inversion procedure used to solve the kernel matrix equation. To ensure that SKA is able to match the sketch size of alternative methods under the compute budget of $2n$, we augment SKA with random samples when necessary:

- If the target sketch size is smaller than $n^{\frac{1}{3}}$ ($s < n^{\frac{1}{3}}$), we use SKA to produce a sketch of size s from a subsample of n/s data points.
- For $s > n^{\frac{1}{3}}$, we use SKA to produce a sketch of size $s_c = n^{\frac{1}{3}}$ from a subsample of $n_c = n^{\frac{2}{3}}$ data points. We match the difference in sketch size by taking an additional $s - s_c$ random samples from the remaining $n - n_c$ data points that were not used for the SKA sketch. The final estimate is a weighted average between the SKA sketch and the uniform sketch: $\frac{1}{s_c}$ for SKA and $(1 - \frac{1}{s_c})$ for uniform, where the weights are determined by the size of the two sketches.

The modification uses SKA as a form of regularization on random samples. Since SKA iteratively selects points that are farthest away from the current set, the resulting sketch is helpful in predicting the "sparser" regions of the space. These sparser regions, in turn, are the ones that survive in the

Table 4.2: Overview of algorithm complexity and parameter choice for the sketching experiment (n: dataset size, s: sketch size, T: number of hash tables, m: sample size for herding).

Algorithm	Complexity	Parameters
HBS	$O(n'T + s)$	$n' = \frac{2n}{5}, T = 5$
SKA	$O(n_c s_c + s_c^3)$	$s_c = n^{\frac{1}{3}}, n_c = n^{\frac{2}{3}}$
Herding	$O(n_h m + n_h s)$	$m = s, n_h = \frac{n}{s}$

$n^{2/3}$ random sample of the dataset (sub-sampling with probability $n^{-1/3}$), therefore SKA naturally includes points from "sparse" clusters of size $\Omega(n^{1/3})$ in the original set.

Herding. The kernel Herding algorithm (Algorithm 9) first estimates the density of the dataset via random sampling; the sketch is then produced by iteratively selecting points with maximum residual density. The algorithm has a complexity of $O(nm+ns)$, where m stands for the sample size used to produce the initial density estimates. To keep Herding under the same $2n$ compute budget, we downsample the dataset to size $n_h = \frac{n}{s}$, and use $m = s$ samples to estimate the initial density. This means that, the larger the sketch size is, the less accurate the initial density estimate is. As a result, we observe degrading performance at larger sketch sizes $s = \Omega(\sqrt{n})$.

Algorithm 8 Sparse Kernel Approximation (SKA)

Input: set P, kernel K, size s.
$S = \{x_1, \ldots, x_s\} \leftarrow$ Greedy-kcenter(P, s)
$K \in \mathbb{R}^{s \times s}$ with $K_{ij} \leftarrow k(x_i, x_j)$ for $x_i, x_j \in S$.
$y \in \mathbb{R}^s$ with $y_i \leftarrow \mathrm{KDE}_P^w(x_i)$ for $x_i \in S$.
Let $\hat{w}$ be a solution to $K\hat{w} = y$.
Output: $(S, \hat{w})$

Algorithm 9 Approximate Kernel Herding (AKH)

Input: set P, kernel K, size s, samples m.
for $i = 1, \ldots, |P|$ **do**
    $P_i \leftarrow \mathrm{Random}(P, m)$. ▷ random set of m points
    $d_i \leftarrow \mathrm{KDE}_{P_i}(x_i)$ ▷ estimate of the density
end for
$S_0 \leftarrow \emptyset$ ▷ initialization
for $t = 1, \ldots, s$ **do**
    $j^* \leftarrow \arg\max_{i \in [n]} \{d_i - \mathrm{KDE}_{S_{t-1}}(x_i)\}$ ▷ greedy
    $S_t \leftarrow S_{t-1} \cup \{x_j^*\}$ ▷ add point to the set
end for
Output: $(S_s, \frac{1}{s}\mathbf{1}_s))$ ▷ return the sketch

Figure 4.7 reports the relative error achieved by different sketching procedures on random queries and low-density queries on real-world datasets. Uniform, SKA and HBS achieve similar mean error

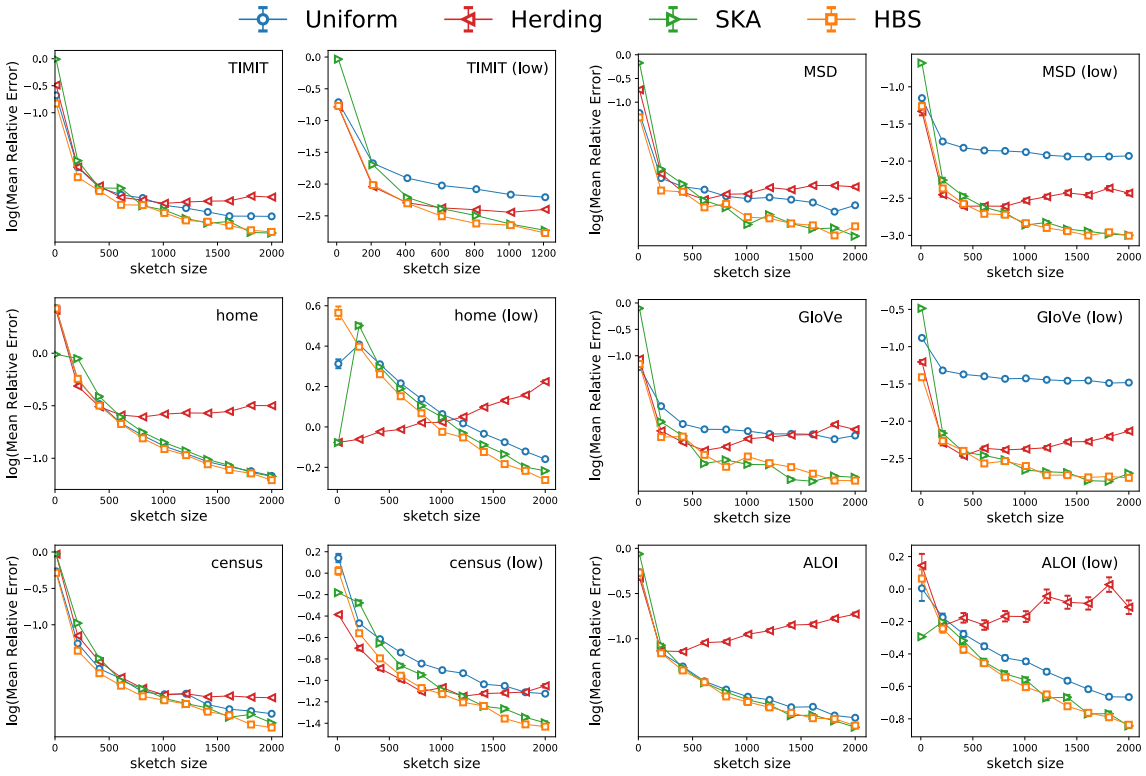


Figure 4.7: Relative error of KDE queries using different sketching procedures on real-world datasets.

on random queries, while the latter two have improved performance on low-density queries, with HBS performing slightly better than SKA. By design, HBS has similar performance with random sampling on average, but performs better on relatively "sparse" regions due to the theoretical guarantee that buckets with weight at least τ are sampled with high probability. Due to the "regularization" phenomenon described above, we observe that SKA+Uniform has improved performances on low-density data points, and consistently outperforms uniform sampling in the experiments. In a few datasets, SKA experiences an initial performance degradation; this tends to happen when we first start to introduce random samples into the SKA sketch. Finally, Herding's performance tends to follow a "U" shape: when sketch sizes are small, the sketch does not have enough capacity to capture more information of the original dataset; when sketch sizes are large, the density estimates are less accurate as a result of increased downsampling. Kernel Herding is competitive only for a small number of points in the sketch.

Finally, in Table 4.3, we report on the estimated precomputatation runtime reduction enabled by HBS for the density estimation results reported in Table 4.1. The estimates are calculated by dividing the number of data points hashed according to the original HBE procedure by the number of data points hashed after enabling HBS.

Table 4.3: Estimated overhead reduction enabled by HBS.

	census	TMY3	TIMIT	SVHN	covertype	MSD	GloVe	ALOI
Reduction (est.)	958×	755×	821×	659×	554×	607×	595×	303×

4.6 Discussion

4.6.1 Related Work

Fast Multipole Methods and Dual Tree Algorithms. This problem of KDE evaluation historically was first studied in low dimensions ($d \leq 3$) in the scientific computing literature, resulting in the Fast Multipole Method (FMM) [138]. This influential line of work is based on hierarchical spatial decomposition and runs in $\tilde{O}(2^d)$ time per evaluation point. Motivated by applications in statistical learning [136], the problem was revisited in 2000's and analogues of the Fast Multipole Method [137], known as Dual Tree Algorithms [199], were proposed that aimed to capture latent underlying low-dimensional structure [261]. Unfortunately, when such low-dimensional structure is absent, these methods in general have near-linear $O(n^{1-o(1)})$ runtime. Thus, under general assumptions, the only method that was known [198] to provably accelerate kernel evaluation in high dimensions was simple uniform random sampling (RS) requiring $O(\frac{1}{\epsilon^2}\frac{1}{\mu})$ samples where $\mu := \mathrm{KDE}_P^u(q)$.

Hashing-Based-Estimators. Aiming to improve upon simple random sampling, [300] and independently [63] introduced a class of methods that use hashing to perform importance sampling to construct unbiased estimators for KDE at arbitrary query points $q \in \mathbb{R}^d$. [63] showed that this approach offers provable improvements compared to random sampling. In particular, for the Generalized t-student, Laplace (Exponential) and Gaussian kernels, the authors showed that one can estimate the density at any point using only $O(\frac{1}{\epsilon^2}\frac{1}{\sqrt{\mu}})$ samples. Recent advances have also extended such techniques to more general kernels [28, 64]. There is also work that applies hashing to get practical improvements to problems besides kernel evaluation such as outlier detection [209], gradient estimation [75] and non-parametric clustering [210].

Approximate Skeletonization. A different line of work aimed at addressing the deficiencies of FMM and Dual-Tree methods is that of ASKIT [219]. This method also produces a hierarchical partition of points but then uses linear algebraic techniques to approximate the contribution of points to each part by a combination of uniform samples and nearest neighbors. This makes the method robust to the dimension and mostly dependent on the number of points.

Core-sets. The problem of sketching the KDE *for all points* has been studied under the name of Core-sets or ϵ-samples [251]. The methods are very effective in low dimensions [342] $d \leq 3$, but become impractical to implement in higher dimensions for large data-sets due to their computational requirements $\Omega(n^2)$ (e.g. [77]). For an up to date summary see the recent paper [252].

4.6.2 Future Directions

Our work shows promising results towards translating theoretical advances into a practical system with performance improvements on real-world datasets. To do so, we improved several aspects of the HBE, such as reducing the constant factor of the sampling algorithm, reducing precomputation overheads with sketching, and generalizing the analysis beyond worst-case scenarios. However, many opportunities remain for future research.

First, while our cost-based optimizer can accurately estimate the sampling efficiency of HBE and RS at a small cost, better sampling efficiency does not directly translate to better query time. We have empirically observed that HBE queries are a constant time slower than RS queries due to the additional complexity of the algorithm. The cost-based optimizer could be augmented with machine- and dataset-specific parameters that account for the performance gap in wall-clock time.

Second, while our precomputed hash tables naturally allow insertion operations, supporting data updates is not free. However, if we keep getting new data points generated from a significantly different underlying distribution than existing ones, we would eventually need to update the LSH parameters to account for the differences. It is unclear how to handle insertions from out-of-distribution points without requiring rebuilding hash tables from scratch.

Finally, our work focuses on the evaluation/inference phase of kernel density estimation, where the weights of each data point are fixed. However, in the training phase, the weights can change from iteration to iteration. Rebuilding hash tables every iteration is too expensive, so extending HBE to the training phase requires additional considerations.

Part II

Improving Human Efficiency

Chapter 5

ASAP: Automatic Smoothing in Time Series Visualization

This chapter presents a novel visualization operator ASAP, designed to focus end-users' attention on large-scale trends and deviations in time series via automatic smoothing. Application authors, site operators, and DevOps engineers collect and store large-scale volumes of time-stamped measurements from infrastructure and applications (i.e., time series) to perform monitoring, health checks, alerts, and analyses of unusual events such as failures [44, 160]. Despite the prevalence of monitoring systems that range from on-premise software, including Ganglia [125], Graphite [135], Prometheus [258], and Facebook Gorilla [243], to cloud services including DataDog [92], New Relic [235], AWS CloudWatch [81], Google Stackdriver [304], and Microsoft Azure Monitor [224], the effective visualization of time series remains a challenge.

Through conversations with engineers who use time series data and databases in cloud services, social networking, industrial manufacturing, electrical utilities, and mobile applications, we learned that many production time series visualizations (i.e., dashboards) simply display raw data streams as they arrive. The engineers reported that this display of raw data can be a poor match for production scenarios involving data exploration and debugging. As data arrives in increasing volumes, even small-scale fluctuations in data values can obscure overall trends and behaviors. For example, an electrical utility employs two staff to perform 24-hour monitoring of the generators. It is critical that these staff quickly identify any systematic shifts in the generator metrics on their monitoring dashboards, even those that are "sub-threshold" with respect to a critical alarm. Unfortunately, such sub-threshold events are easily obscured by short-term fluctuations in the visualization.

The resulting challenge in time series visualizations at scale is presenting the *appropriate* plot that prioritizes users' attention towards significant deviations. The time series depicted in Figure 5.1

[1]Here and later in this section, we depict z-scores [192] instead of raw values. This choice of visualization provides a means of normalizing the visual field across plots while still highlighting large-scale trends.

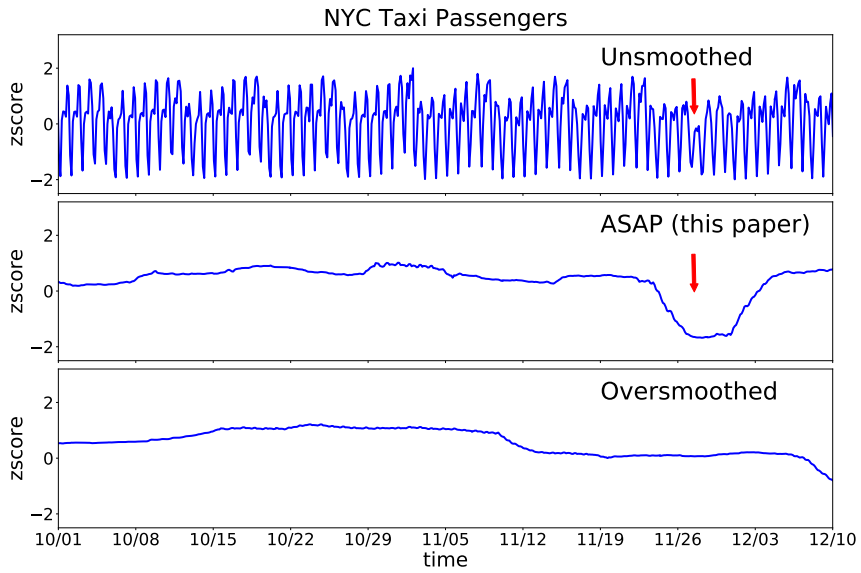


Figure 5.1: Normalized number of NYC taxi passengers over 10 weeks.[1] From top to bottom, the three plots show the hourly average (unsmoothed), the weekly average (smoothed) and the monthly average (oversmoothed) of the same time series. The arrows point to the week of Thanksgiving (11/27), when the number of passengers dips. This phenomenon is most prominent in the smoothed plot produced by ASAP, the subject of this section.

illustrates this challenge. The top plot displays the raw data: an hourly average of the number of New York City taxi passengers over 75 days in 2014 [197]. The daily fluctuations of taxi volume dominate the visual field, obscuring a significant long-term deviation: the number of taxi passengers experienced a sustained dip during the week of Thanksgiving. The ideal solution would be to smooth the local fluctuations to highlight this deviation in the visualization (Figure 5.1, middle). However, if smoothed too aggressively, the visualization may hide this trend entirely (Figure 5.1, bottom).

In this work, we address the challenge of prioritizing analysts' attention in time series visualizations using a simple strategy: smooth time series visualizations as much as possible while preserving large-scale deviations. This raises two key questions. First, how can we quantitatively assess the quality of a given visualization in removing small-scale variations and highlighting significant deviations? Second, how can we use such quantitative metrics to produce high-quality visualizations at scale? We answer both questions through the design of a new time series visualization operator called ASAP (Automatic Smoothing for Attention Prioritization). ASAP quantifiably improves end-user accuracy and speed in identifying significant deviations in time series, and is optimized to execute at scale.[2]

To address the first question of quantitatively assessing visualization quality, we combine two statistics. First, we measure the smoothness of a time series visualization via the *variance of first*

[2]The demo and code are available at `http://futuredata.stanford.edu/asap/`

differences [66], or the variation of differences between consecutive points in the series. Applying a moving average of increasing length reduces this variance and smooths the plot. However, as illustrated by Figure 5.1, it is possible to oversmooth and obscure the trend entirely. Therefore, to prevent oversmoothing, we introduce a constraint based on preserving the *kurtosis* [96]—a measure of the "outlyingness" of a distribution—of the original time series, preserving its structure. Incidentally, this kurtosis measure can also determine when *not* to smooth (e.g., if a series has a few well-defined outlying regions). We demonstrate the utility of this combination of smoothness measure and constraint via two user studies. Compared to displaying raw data, smoothing time series visualizations using these metrics improves users' accuracy in identifying anomalies by up to 38.4% and decreases response times by up to 44.3%.

Using these metrics, ASAP automatically selects smoothing parameters on users' behalf, producing the smoothest visualization that still retains large-scale deviations. Given a window of time to visualize (e.g., the past 30 minutes of a time series), ASAP selects and applies an appropriate smoothing parameter to the target series. Unlike existing smoothing techniques that are designed to produce visually indistinguishable representations of the original signal (e.g., [173, 286]), ASAP is designed to "distort" visualizations (e.g., by removing local fluctuations) to highlight key deviations (e.g., as in Figure 5.1) and prioritize analysts' attention [30].

There are three main challenges to enabling this efficient, automatic smoothing. First, our target workloads exhibit large data volumes—up to millions of events per second—so ASAP must produce legible visualizations despite high volumes. Second, to support interactive use, ASAP must render quickly. As we demonstrate, an exhaustive search over smoothing parameters for 1 million points requires over an hour, yet we target sub-second response times. Third, appropriate smoothing parameters may change over time: a high-quality parameter choice for one time period may oversmooth or undersmooth in another. Therefore, ASAP must adapt its smoothing parameters in response to changes in the streaming time series.

To address these challenges, ASAP combines techniques from stream processing, user interface design, and signal processing. First, to scale to large volumes, ASAP pushes constraints regarding the target end-user display into its design. ASAP exploits the fact that its results are designed to be displayed in a fixed number of pixels (e.g., a maximum of 1334 pixels at a time on the iPhone 7), and uses target resolution as a natural lower bound for the parameter search; choosing parameters that would result in a resolution greater than the target display size is rarely beneficial. Accordingly, ASAP pre-aggregates the data, thus reducing the search space. Second, to further improve the rendering time, ASAP prunes the search space by searching for period-aligned time windows (i.e., a time lag with high autocorrelation) for periodic data and performing binary search for aperiodic data. Our findings demonstrate that this search strategy leads to smooth aggregated series both analytically and empirically. Third, to quickly respond to changes in fast-moving time series, ASAP avoids recomputing smoothing parameters from scratch upon the arrival of each new data point.

Instead, ASAP reuses computation and re-renders visualizations on human-observable timescales.

ASAP achieves its goals of efficient and automatic smoothing by treating visualization properties, including end-user display constraints and limitations of human perception as critical design considerations. As we empirically demonstrate, this co-design yields useful results, quickly and without manual tuning. We have implemented ASAP as a time series explanation operator in the MacroBase fast data engine [29], and as a JavaScript library. The resulting ASAP prototypes demonstrate order-of-magnitude runtime improvements over alternative search strategies while producing high-quality smoothed visualizations.

In summary, this work makes the following contributions:

- ASAP, the first stream processing operator for automatically smoothing time series to reduce local variance (i.e., minimize roughness) while preserving large-scale deviations (i.e., preserving kurtosis) in visualizations.
- Three optimizations for improving ASAP's execution speed that leverage 1) target device resolution in pre-aggregation, 2) autocorrelation to exploit periodicity, and 3) partial materialization for streaming updates.
- A quantitative evaluation demonstrating ASAP's ability to improve user accuracy and response time and deliver order-of-magnitude performance improvements.

The remainder of this chapter proceeds as follows. Section 5.1 provides additional background regarding our target use cases as well as an overview of ASAP's architecture and problem statement. Section 5.2 describes ASAP's basic search strategy as well as its optimizations for the batch and streaming execution modes. Section 5.3 evaluates ASAP's visualization quality through two user studies, and Section 5.4 evaluates ASAP's performance on a range of synthetic and real-world datasets. Section 5.5 discusses related work, alternative design choices and ASAP's real-world usage and deployments.

5.1 Overview and Problem Statement

5.1.1 Architecture and Usage

ASAP provides analysts and system operators an effective and efficient means of highlighting large-scale deviations in time series visualizations. In this section, we describe ASAP's usage and architecture, illustrated via two additional case studies.

Given an input time series (i.e., set of temporally ordered data points) and target interval for visualization (e.g., the last twelve hours of data), ASAP returns a transformed, smoothed time series (e.g., also of twelve hours, but with a smoothing function applied) for visualization. In the streaming setting, as new data points arrive, ASAP continuously smooths each fixed-size time

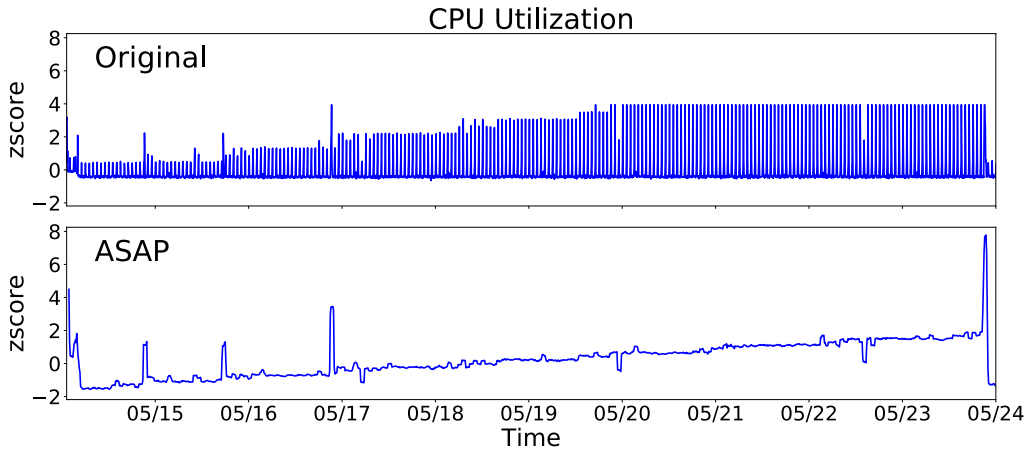


Figure 5.2: Server CPU usage across a cluster over ten days [197], visualized via a 5 minute average (raw) and an hourly average (via ASAP). The CPU usage spike around May 24th is obscured by frequent fluctuations in the raw time series.

interval, producing a sequence of smoothed time series. Thus, ASAP acts as a transformation over fixed-size sliding windows over a single time series. When ASAP users change the range of time series to visualize (e.g., via zoom-in, zoom-out, scrolling), ASAP re-renders its output in accordance with the new range. For efficiency, ASAP also allows users to specify a target display resolution (in pixels) and a desired refresh rate (in seconds).

ASAP can run either client-side or server-side. For easy integration with web-based front-ends, ASAP can execute on the client; we provide a JavaScript library for doing so. However, for resource-constrained clients, or for servers with a large number of visualization consumers, ASAP can execute on the server, sending clients the smoothed stream; this is the execution mode that MacroBase [29] adopts, and MacroBase's ASAP implementation is portable to existing stream processing engines.

ASAP acts as a modular tool in time series visualization. It can ingest and process raw data from time series databases such as InfluxDB, as well as from visualization clients such as plotting libraries and frontends. For example, when building a monitoring dashboard, a DevOps engineer could employ ASAP and plot the smoothed results in his metrics console, or, alternatively, overlay the smoothed plot on top of the original time series. ASAP can also *post-process* outputs of time series analyses including motif discovery, anomaly detection, and clustering [185,187,205,344]: given a single time series as output from each of these analyses, ASAP can smooth the time series prior to visualization.

To further illustrate ASAP's potential uses in prioritizing attention in time series, we provide two additional case studies cases below, and additional examples of raw time series and their smoothed counterparts in Appendix C.2:

Application Monitoring. An on-call application operator is paged at 4AM due to an Amazon

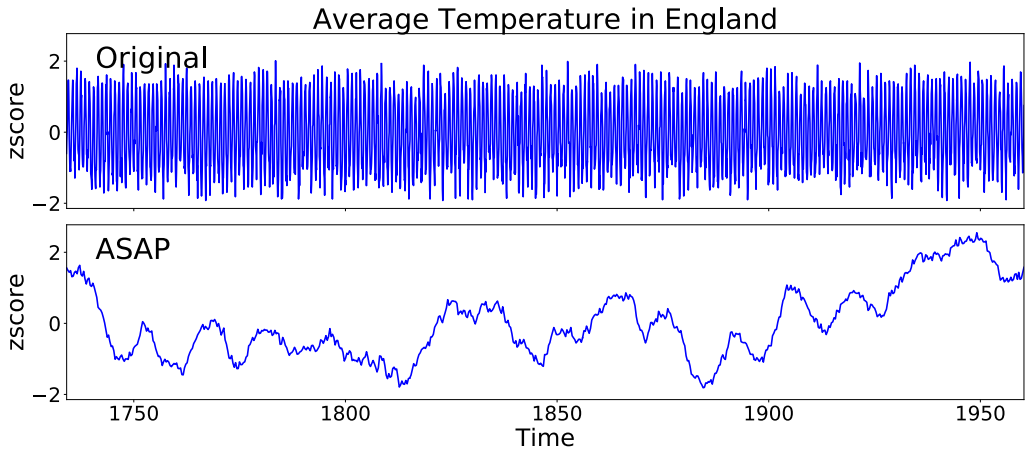


Figure 5.3: Temperature in England from 1723 to 1970 [164], visualized via a monthly average (raw) and 23-year average (via ASAP). Fluctuations in the raw time series obscure the overall trend.

CloudWatch alarm on a sudden increase in CPU utilization on her Amazon Web Service cloud instances. After reading the alert message, she accesses her cluster telemetry plots that include CPU usage over the past ten days on her smartphone to obtain a basic understanding of the situation. However, the smartphone's display resolution is too small to effectively display all 4000 readings; as a result, the lines are closely stacked together in the plot, making CPU usage appear stable (Figure 5.2, top).[3] Unable to obtain useful insights from the plot, the operator must rise from bed and begin checking server logs manually to diagnose the issue. If she were to instead apply ASAP, the usage spike around May 24th would no longer be hidden by noise.

Historical Analyses. A researcher interested in climate change examines a data set of monthly temperature in England over 200 years. When she initially plots the data to determine long-term trends, her plot spills over five lengths of her laptop screen.[4] Instead of having to scroll to compare temperature in the 1700s with the 1930s, she decides to plot the data herself to fit the entire time series onto one screen. Now, in the re-plotted data (Figure 5.3, top), seasonal fluctuations each year obscure the overall trend. Instead, if she were to instead use ASAP, she would see a clear trend of rising temperature in the 1900s (Figure 5.3, bottom).

[3]This plot is inspired by an actual use case we encountered in production time series from a large cloud operator; high frequency fluctuations in the plot made it appear that a server was behaving abnormally, when in fact, its overall (smoothed) behavior was similar to others in the cluster.

[4]This is not a theoretical example; in fact, the site from which we obtained this data [164] plots the time series in a six-page PDF. This presentation mode captures fine-grained structure but makes it difficult to determine long-term trends at a glance, as in Figure 5.3.

5.1.2 Problem Definition

Next, we formally present ASAP's problem statement. We first introduce the two key metrics that ASAP uses to assess the quality of smoothed visualizations as well as its smoothing function. We subsequently cast ASAP's parameter search as an optimization problem.

Roughness Measure

As we have discussed, noise and/or frequent fluctuations can distract users from identifying large-scale trends in time series visualizations. Therefore, to prioritize user attention, we wish to *smooth as much as possible while preserving systematic deviations*. We first introduce a metric to quantify the degree of smoothing.

Standard summary statistics such as mean and standard deviation alone may not suffice to capture a time series's visual smoothness. For example, consider the three time series in Figure 5.4: a jagged line (series A), a slightly bent line (series B), and a straight line (series C). These time series appear different, yet all have a mean of zero and standard deviation of one. However, series C looks "smoother" than series A and series B because it has a constant slope. Put another way, the differences between consecutive points in series C have smaller variation than consecutive points in series A and B.

To formalize this intuition, we define the roughness (i.e., inverse "smoothness," to be minimized) of a time series as the standard deviation of the differences between consecutive points in the series. The smaller the variation of the differences, the smoother the time series. Formally, given time series $X = \{x_1, x_2, ..., x_N\}$, $x_i \in \mathbb{R}$, we adopt the concept of the *first difference* series [66] as:

$$\Delta X = \{\Delta x_1, \Delta x_2, ...\} \quad s.t. \quad \Delta x_i = x_{i+1} - x_i, i \in \{1, 2, ..., N-1\}$$

Subsequently, we can define the roughness of time series X as the standard deviation of the first difference series:

$$\text{roughness}(X) = \sigma(\Delta X)$$

This use of variance of differences is closely related to the concept of a *variogram* [89], a commonly-used measure in spatial statistics (especially geostatistics) that characterizes the spatial continuity (or *surface roughness*) of a given dataset. By this definition, the roughness of the three time series in Figure 5.4 are 2.04, 0.4, and 0, respectively. Note that a time series will have a roughness value of 0 if and only if the corresponding plot is a straight line (like series C). Specifically, a roughness value of 0 implies the differences between neighboring points are identical and therefore the plot corresponding to the series will have a constant slope, resulting in a straight line.

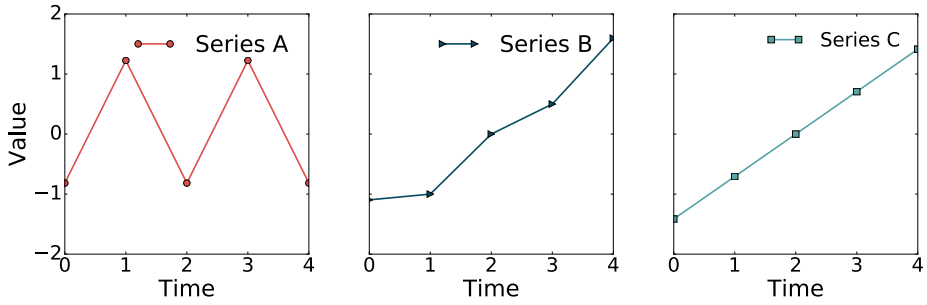


Figure 5.4: Three time series that appear visually distinct yet all have mean of zero and standard deviation of one. This example illustrates that standard summary statistics such as mean and standard deviation can fail to capture the visual "smoothness" of time series.

Preservation Measure

Per the above observation, if we simply minimize roughness, we will produce plots that approximate straight lines. In some cases, this is desirable; if the overall trend is a straight line, then removing noise may, in fact, result in a straight line. However, as our examples in Section 5.1 demonstrate, many meaningful trends are not accurately represented by straight lines. As a result, we need a measure of "trend preservation" that captures how well we are preserving large-scale deviations within the time series.

To quantify how well we are preserving large deviations in the original time series, we measure the distribution *kurtosis* [96]. Kurtosis captures "tailedness" of the probability distribution of a real-valued random variable, or how much mass is near the tails of the distribution. More formally, given a random variable X with mean μ and standard deviation σ, kurtosis is defined as the fourth standardized moment:

$$\text{Kurt}[X] = \frac{\mathbb{E}[(X-\mu)^4]}{\mathbb{E}[(X-\mu)^2]^2}$$

Higher kurtosis means that more of the variance is contributed by rare and extreme deviations, instead of more frequent and modestly sized deviations [327]. For reference, the kurtosis for univariate normal distribution is 3. Distributions with kurtosis less than 3, such as the uniform distribution, produce fewer and less extreme outliers compared to normal distributions. Distributions with kurtosis larger than 3, such as the Laplace distribution, have heavier tails compared to normal distributions. Figure 5.5 illustrates two time series sampled from the normal and Laplace distribution discussed above. Despite having the same mean and variance, kurtosis captures the two series' difference in tendency to produce outliers.

To prevent oversmoothing large-scale deviations in the original time series, we compare the kurtosis of the time series before and after applying the smoothing function. If the kurtosis of the original series is greater than or equal to the smoothed series, then the proportion of values that significantly deviate in the smoothed series is no smaller than the proportion in the original series.

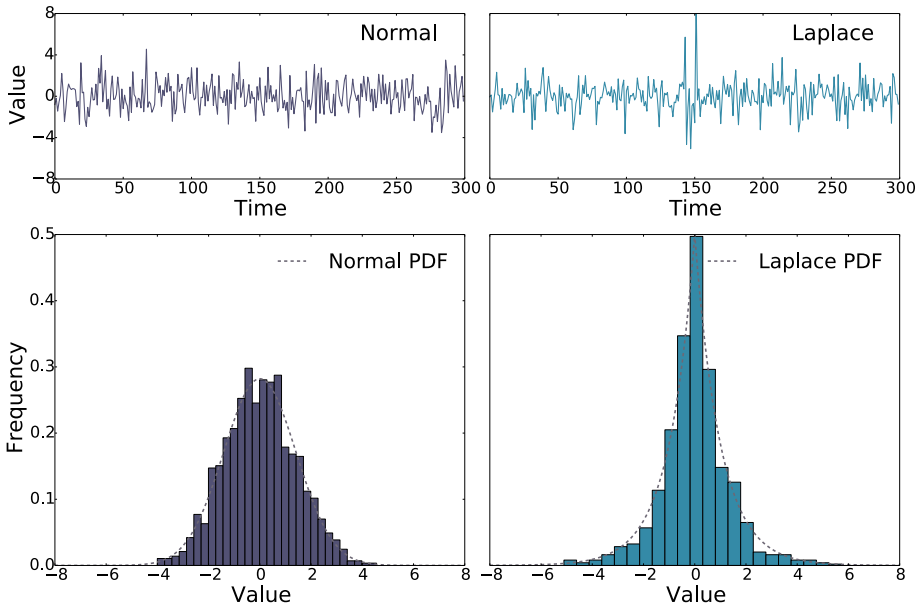


Figure 5.5: Time series and histograms sampled from a normal distribution (left) and a Laplace distribution (right). Despite having the same mean (0) and variance (2), the Laplace series includes a few large deviations, while the normal includes a large number of moderate deviations. The difference in tendency to produce outliers is captured by kurtosis: the normal distribution has a kurtosis of 3, while the Laplace distribution has a kurtosis of 6.

If smoothing is effective, then the smoothing will "concentrate" the values around regions of large deviation (i.e., significant shifts from the mean) and therefore highlight these deviations.

If the original time series only contains a few extreme outliers, the smoothing is likely to only average out the deviations, which we also account for in our parameter selection procedure. For example, consider a time series with all but one point in the range $[-1, 1]$ and a single outlying point that has a value of 10. This outlier may be the most important piece of information that users would like to highlight in the time series, so applying a simple moving average only decreases the extent of this deviation (i.e., the kurtosis of the smoothed time series decreases). The kurtosis preservation constraint thus ensures we leave the original time series unsmoothed.

Smoothing Function

Given our roughness and preservation measure, we wish to smooth our time series as much as possible (i.e., minimizing roughness) while preserving large-scale deviations (i.e., preserving kurtosis). To perform the actual smoothing, we need a smoothing function.

In this paper, we focus on simple moving average (SMA) as the smoothing function. Three reasons motivate this choice. First, SMA is well studied in the stream processing literature, with several existing techniques for efficient execution and incremental maintenance [201]. We adopt these

techniques, while using roughness and preservation metrics as a means of automatically tuning SMA parameters for visual effect. Second, SMA is also well studied in the signal processing community. Statistically, the moving average is optimal for recovering the underlying trend of the time series when the fluctuations about the trend are normally distributed [298], despite its light computational footprint and conceptual simplicity compared to alternatives. Third, we experimented with several alternatives including the MinMax aggregation, the Fourier transform [297], and the Savitzky-Golay filter [276]; SMA had fewer parameters to tune and proved more effective at smoothing per our target metrics. We include a visual comparison in [267].

Given input $w \in \mathbb{N}$, SMA averages every sequential set of w points in the original time series X to produce one point in the smoothed series Y. We can express SMA as:

$$\text{SMA}(X, w) = \{y_1, ...y_{N-w}\} \quad s.t. \quad y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}$$

When applying SMA over data streams with a sliding window, users can adjust its *window size* (number of points in each window) and *slide size* (distance between neighboring windows) parameters. In time series visualization, slide size determines the sampling frequency of the original time series and, therefore, the number of distinct, discrete data points in the smoothed plot. In this work, we focus on automatically selecting a window size for a given slide size. Instead of tuning slide size, we employ a policy that sets slide size according to the desired number of points (i.e., pixels) in the final visualization (i.e., $\frac{\text{\# original points}}{\text{\#desired points}}$). Increasing the slide size beyond this threshold results in fewer data points than specified in the smoothed visualization, and decreasing the slide sizes results in a smoothed time series with more data points than available display resolution. Therefore, we found that varying the slide size did not dramatically improve visualization quality.

ASAP Problem Statement

Given our roughness and preservation measures and smoothing function, we present ASAP's problem statement as follows:

PROBLEM. *Given time series* $X = \{x_1, x_2, ..., x_N\}$, *let* $Y = \{y_1, y_2, ..., y_{N-W}\}$ *be the smoothed series of* X *obtained by applying a simple moving average with window size* w *(i.e.,* $y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}$*). Find window size* $\hat{w}$ *where:*

$$\hat{w} = \arg\min_{w} \sigma(\Delta(Y)) \quad s.t. \quad \text{Kurt}[Y] \geq \text{Kurt}[X]$$

That is, we wish to reduce roughness in a given time series as much as possible by applying a sliding window average function to the data while preserving kurtosis.

5.2 Implementation and Optimizations

In this section, we describe ASAP's core search strategy and optimizations for solving the problem of smoothing parameter selection. We first focus on smoothing a single, fixed-length time series, beginning with a walkthrough of a strawperson solution and an analysis of the problem dynamics under a simple, IID distribution (Section 5.2.1). Using the insights from this analysis, we develop a pruning optimization based on autocorrelation (Section 5.2.2). We then introduce a pixel-aware optimization that greatly reduces the input space via preaggregation (Section 5.2.3). Finally, we discuss additional optimizations for the streaming setting (Section 5.2.4).

5.2.1 Strawperson Solution and IID Analysis

Strawperson Solution. We could exhaustively search all possible window lengths and return the one that gives the smallest roughness measure while satisfying the kurtosis constraint. For each candidate window length, we need to smooth the series and evaluate the roughness and kurtosis. Each of these computations requires linear time ($O(N)$). However, there are also many candidates to evaluate: for a time series of size N, we may need to evaluate up to N possible window lengths, resulting in a total running time of $O(N^2)$. As we illustrate empirically in Section 5.4, in the regime where N is even modestly large, this computation can be prohibitively expensive.

We might consider improving the runtime of this exhaustive search by performing grid search via a sequence of larger step sizes, or by performing binary search. However, as we will demonstrate momentarily, the roughness metric is not guaranteed to be monotonic in window length and therefore, the above search strategies may deliver poor quality results.

Basic IID Analysis. To develop a more efficient search strategy, we first consider how window length affects the roughness and kurtosis of the smoothed series.

Consider a time series $X : \{x_1, x_2, ..., x_N\}$ consisting of samples drawn identically independently distributed (IID) from some distribution with mean μ and standard deviation σ. After applying a moving average of window length w, we obtain the smoothed series:

$$Y = \text{SMA}(X, w), \quad y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}, i \in \{1, 2, ..., N-w\}$$

We denote the first difference series as $\Delta Y = \{\Delta y_1, \Delta y_2, ...\}$, where

$$\Delta y_i = y_{i+1} - y_i = \frac{1}{w}\sum_{j=0}^{w-1}(x_{i+j+1} - x_{i+j}) = \frac{1}{w}(x_{i+w} - x_i)$$

For convenience, we also denote the first $N-w$ points of X as $X_f = \{x_1, x_2, ..., x_{N-w}\}$ and the last $N-w$ points of X as $X_l = \{x_{w+1}, x_{w+2}, ..., x_N\}$. Then $\Delta Y = \frac{1}{w}(X_l - X_f)$, and roughness of the

smoothed series Y can be written as:

$$\text{roughness}(Y) = \sigma(\Delta Y) = \frac{1}{w}\sqrt{\text{var}(X_f) + \text{var}(X_l) - 2\text{cov}(X_l, X_f)} \tag{5.1}$$

Since each x_i is drawn IID from the same distribution, we have $\text{var}(X_f) = \text{var}(X_l) = \sigma^2$ and $\text{cov}(X_f, X_l) = 0$. Substituting in Equation 5.1 we obtain:

$$\text{roughness}(Y) = \frac{\sqrt{2}\sigma}{w} \tag{5.2}$$

Therefore, for IID data, roughness linearly decreases with increased window size. Further, the kurtosis of random variable S, defined as the sum of independent random variables $R_1, ..., R_n$, is

$$\text{Kurt}[S] - 3 = \frac{1}{(\sum_{j=1}^{n} \sigma_j^2)^2} \sum_{i=1}^{n} \sigma_i^4 (\text{Kurt}[R_i] - 3) \tag{5.3}$$

where σ_i is the standard deviation of the random variable R_i. In our case, Y is the sum of w IID random variables X [236]. Thus, Equation 5.3 simplifies to

$$\text{Kurt}[Y] - 3 = \frac{\text{Kurt}[X] - 3}{w}. \tag{5.4}$$

Therefore, for IID series drawn from distributions with initial kurtosis less than 3, kurtosis monotonically increases with window length and for series drawn from distributions with initial kurtosis larger than 3, kurtosis monotonically decreases.

In summary, these results indicate that for IID data, we can simply search for the largest window length that satisfies kurtosis constraint via binary search. Specifically, given a range of candidate window lengths, ASAP applies SMA with window length that is in the middle of the range. If the resulting smoothed series violates the kurtosis constraint, ASAP searches the smaller half of the range; otherwise, ASAP searches the large half. This binary search routine is justified because the roughness of the smoothed series monotonically decreases with window length (Equation 5.2), and the kurtosis of the smoothed series monotonically decreases with window length or achieves its minimum at window length equals one (Equation 5.4).

However, many time series exhibit temporal correlations, which breaks the above IID assumption. In the remainder of this section, we describe an alternative search strategy that is able to retain the quality of exhaustive search while achieving meaningful speedups by quickly pruning unpromising candidates and by optimizing for the desired pixel density.

5.2.2 Optimization: Autocorrelation-Based Pruning

We have just shown that, for IID data, binary search is accurate, yet many time series are not IID; instead, they are often periodic or exhibit other temporal correlations. For example, many servers and automated processes have regular workloads and exhibit periodic behavior across hourly, daily, or longer intervals.

To measure temporal correlations within a time series, we measure the time series *autocorrelation*, or the similarity of a signal with itself as a function of the time lag between two points [291]. Formally, given a process X whose mean μ and variance σ^2 are time independent (i.e., is a *weakly stationary* process), denote X_t as the value produced by a given run of the process at time t. The lag τ autocorrelation function (ACF) on X is defined as

$$\text{ACF}(X, \tau) = \frac{\text{cov}(X_t, X_{t+\tau})}{\sigma^2} = \frac{\mathbb{E}[(X_t - \mu)(X_{t+\tau} - \mu)]}{\sigma^2}$$

The value of the autocorrelation function ranges from [-1, 1], with 1 indicating perfect correlation, 0 indicating the lack of correlation and -1 indicating anti-correlation.

Autocorrelation and Roughness

As suggested above, we can take advantage of the periodicity in the original time series to prune the search space. Specifically, given the original time series $X : \{x_1, x_2, ..., x_N\}$, and the smoothed series $Y : \{y_1, y_2, ..., y_{N-w}\}$ obtained by applying a moving average of window length w, we show that

$$\text{roughness}(Y) = \frac{\sqrt{2}\sigma}{w}\sqrt{1 - \frac{N}{N-w}\text{ACF}(X, w)} \tag{5.5}$$

for a weakly stationary process X. We provide a full derivation of Equation 5.5 in Appendix C.1.1; however, intuitively, this equation illustrates that window length and autocorrelation both affect roughness. For example, consider a time series recording the number of taxi trips taken over 30-minute intervals. Due to the regularity of commuting routines, this time series exhibits autocorrelation across week-long periods (e.g., a typical Monday is likely to be much more similar to another Monday than a typical Saturday). Furthermore, a rolling weekly average of the number of trips should, in expectation, have a smaller variance than rolling 6-day averages: for example, if people are more likely to take taxis during weekdays than during weekends, then the average from Monday to Saturday should be larger than the average from Tuesday to Sunday. Therefore, window lengths that align with periods of high autocorrelation make the resulting series smoother.

We experimentally validate this relationship on real world data (Appendix C.1.1) and use this relationship to aggressively prune the space of windows to search (Section 5.2.2).

Autocorrelation and Kurtosis

In addition to roughness, we also investigate the impact of temporal correlations on the kurtosis constraint. We start with an example that illustrates how choosing window lengths with high temporal correlation (i.e., autocorrelation) leads to high kurtosis.

Consider a time series (sparkline below, left) consisting of a sine wave with 640 data points. Each complete sine wave is 32 data points long, and in the region from 320th to 336th data point, the peak of the sine wave is taller than usual. When applying a window that are multiples of the period, the smoothed series (sparkline below, right) is zero everywhere except around the region where the peak is higher. The smoothed series in the latter case has higher kurtosis because it only contains one large deviation from the mean. In contrast, applying a moving average with window length that is not a multiple of the period will not highlight this peak.

This example illustrates the case when applying moving average with window lengths aligning with the period of the time series can not only remove periodic behavior from the visualization (therefore highlighting deviations from period to period), but also the kurtosis of the smoothed series is also larger at the periodic window size. In ASAP, we find that, empirically, if a candidate window that is aligned with the time series period does not satisfy the kurtosis constraint, it is rare that a nearby candidate window that is off the period would satisfy the constraint instead; moreover, such a nearby aperiodic window would likely result in a rougher series.

Pruning Strategies

Following the above observations, ASAP adopts the following two pruning strategies. The corresponding pseudocode for ASAP's search is listed in Algorithm 10.

Autocorrelation peaks. To quickly filter out suboptimal window lengths, ASAP searches for windows that correspond to periods of high autocorrelation. Specifically, ASAP only checks autocorrelation *peaks*, which are local maximums in the autocorrelation function and correspond to periods in the time series. For periodic datasets, these peaks are usually much higher than neighboring points, meaning that the corresponding roughness of the smoothed time series is much lower. This is justified by Equation 5.5—all else equal, roughness decreases with the increase of autocorrelation.

Naïvely computing autocorrelation via brute force requires $O(n^2)$ time; thus, a brute force this approach is unlikely to deliver speedups over the naïve exhaustive search for finding window length. However, we can improve the runtime of autocorrelation, to $O(n \log(n))$ time, using two Fast Fourier Transforms (FFT) [254]. In addition to providing asymptotic speedups, this approach also allows us to make use of optimized FFT routines designed for signal processing, in the form of mature software libraries and increasingly common hardware implementations (e.g., DSP accelerators).

Algorithm 10 Search for periodic data

Variables:
X: time series; **candidates**: array of candidate window lengths
acf[**w**]: autocorrelation for w; **maxACF**: maximum autocorrelation peak
opt: a set of states for the current best candidate in the search, including {roughness, wLB, window, largestFeasibleIdx}

```
function UPDATELB(wLB, w)                                   ▷ Update lower bound
    return MAX(wLB, w*sqrt((1-maxACF)/(1-acf[w])))
end function
function ISROUGHER(currentBestWindow, w)                    ▷ Compare roughness
    return sqrt(1-acf[w])/w > sqrt(1-acf[currentBestWindow])/currentBestWindow
end function
function SEARCHPERIODIC(X, candidates, opt)
    N = candidates.length
    for i ∈ {N, N-1, ..., 1} do                             ▷ Large to small
        w = candidates[i]
        if w < opt.wLB then                                 ▷ Lower bound pruning
            break
        end if
        if ISROUGHER(opt.window, w) then                    ▷ Roughness pruning
            continue
        end if
        Y = SMA(X, w)
        if ROUGHNESS(Y) < opt.roughness and
            KURT(Y) ≥ KURT(X) then                          ▷ Kurtosis constraint
            opt.window = w
            opt.roughness = ROUGHNESS(Y)
            opt.wLB = UPDATELB(opt.wLB, w)
            opt.largestFeasibleIdx = MAX(opt.largestFeasibleIdx, i)
        end if
    end for
    return opt
end function
```

Large to small. Since roughness decreases with window length (Equation 5.5, roughness is proportional to $\frac{1}{w}$), ASAP searches from larger to smaller window lengths. When two windows $w_1, w_2 (w_1 < w_2)$ have identical autocorrelation, the larger window will always have lower roughness under SMA. However, when the windows have different autocorrelations a_1, a_2, the smaller window w_1 will only provide lower roughness if $w_1 > w_2\sqrt{\frac{1-a_1}{1-a_2}}$. Moreover, since ASAP only considers autocorrelation peaks as candidate windows, a_1 is no larger than the largest autocorrelation peak in the time series, which we refer to as $maxACF$. Therefore, the smallest window w_1 that is able

Algorithm 11 Batch ASAP

```
function findWindow(X, opt)
    candidates = getAcfPeaks(X)
    opt = searchPeriodic(X, candidates, opt)
    head = max(opt.wLB, candidates[opt.largestFeasibleIdx + 1])
    tail = min(maxWindow, candidates[opt.largestFeasibleIdx + 1])
    opt = binarySearch(X, head, tail, opt)
    return opt.window
end function
```

to produce smoother series than w_2 must satisfy

$$w_1 > w_2\sqrt{\frac{1-a_1}{1-a_2}} > w_2\sqrt{\frac{1-maxACF}{1-a_2}} \tag{5.6}$$

If ASAP finds a feasible window length for smoothing relatively early in the search, it uses Equation 5.6 to prune smaller windows that will not produce a smoother series (updateLB in Algorithm 10). Similarly, once ASAP has a feasible window, it can also prune window candidates whose roughness estimate (via Equation 5.5) is larger than the current best (isRougher in Algorithm 10). In summary, the two pruning rules are complementary: the lower bound pruning reduces the search space from below, eliminating search candidates that are too small; the roughness estimate reduces the search space from above, further eliminating unpromising candidates above the lower bound.

Our pruning strategies exploit temporal correlations, which will be less effective for aperiodic data. However, per our analysis in Section 5.2.1, IID data is better-behaved under simple search. Therefore, ASAP falls back to binary search for aperiodic data. ASAP allows users to optionally specify a maximum window size to consider. Together, the search procedure is listed in Algorithm 11.

5.2.3 Optimization: Pixel-aware Preaggregation

In addition to leveraging statistical properties of the data, ASAP can also leverage perceptual properties of the target devices. That is, ASAP's smoothed time series are designed to be displayed on devices such as computer monitors, smartphones, and tablet screens for human consumption. Each of these target media has a limited resolution; as Table 5.1 illustrates, even high-end displays such as the 2016 Apple iMac 5K are limited in horizontal resolution to 5120 pixels, while displays such as the 2016 Apple Watch contain as few as 272 pixels. These pixel densities place restrictions on the amount of information that can be displayed in a plot.

ASAP is able to leverage these limited pixel densities to improve search time. Specifically, ASAP avoids searching for window lengths that would result in more points than pixels supported by the target device. For example, a datacenter server may report CPU utilization metrics every second (604,800 points per week). If an operator wants to view a plot of weekly CPU usage on her 2016 Retina MacBook Pro, she will only be able to see a maximum of 2304 distinct pixels as supported

Table 5.1: Popular devices and search space reduction achieved via pixel-aware preaggregation for a time series with 1M points.

Device	Resolution	Reduction on 1M pts
38mm Apple Watch	272 x 340	3676x
Samsung Galaxy S7	1440 x 2560	694x
13" MacBook Pro	2304 x 1440	434x
Dell 34 Curved Monitor	3440 x 1440	291x
27" iMac Retina	5120 x 2880	195x

by the display resolution. If ASAP smooths using a window smaller than 262 seconds (i.e., $\frac{604,800}{2304}$), the resulting plot will contain more points than pixels on the operator's screen (i.e., to display all information in the original time series, the slide size must be no larger than window length). As a result, this *point-to-pixel* ratio places a lower bound on the window length that ASAP should search. In addition, the point-to-pixel ratio is also a useful proxy for the granularity of information content contained in a given pixel. While one could search for window lengths that correspond to sub-pixel boundaries, in practice, we have found that searching for windows that are integer multiples of the point-to-pixel ratio suffices to capture the majority of useful information in a plot. We provide an analysis in Appendix C.1.2 and empirically demonstrate these phenomena in Section 5.4.2.

Combined, these observations yield a powerful optimization for ASAP's search strategy. Given a target display resolution (or desired number of points for a plot), ASAP pushes this information into its search strategy by only searching windows that are integer multiples of point-to-pixel ratio. To implement this efficiently, ASAP preaggregates the data points according to groups of size corresponding to the point-to-pixel ratio, then proceeds to search over these preaggregated points. With this preaggregation, ASAP's performance is not dependent on the number of data points in the original time series but instead depends on the target resolution of the end device. As a result, in Section 5.4, we evaluate ASAP's performance over different target resolutions and demonstrate scalability to millions of incoming data points per second.

5.2.4 Optimization: Streaming ASAP

ASAP is designed to process streams of time series and update plots as new data arrives. Here, we describe how ASAP efficiently operates over data streams by combining techniques from traditional stream processing with constraints on human perception.

Basic Operations. As new data points arrive, ASAP must update its smoothing parameters to accommodate changes in the trends, such as periodicity. As in Section 5.2.3, in the streaming setting, we can preaggregate data as it arrives according to the point-to-pixel ratio. However, as data transits the duration of time ASAP is configured to smooth (e.g., the last 30 minutes of readings), ASAP must remove outdated points from the window. To manage this intermediate state, ASAP adapts techniques from streaming processing that sub-aggregate input streams for performance gain. That

Algorithm 12 Streaming ASAP

Variables: X: preaggregated time series; **interval**: refresh interval

```
function checkLastWindow(X, opt)
    Y = SMA(X, opt.window)
    if kurt(Y) ≥ kurt(X) then
        update roughness and wLB for opt
    else
        re-initialize opt
    end ifreturn opt
end function
function updateWindow(X, interval)
    while True do
        collect new data points until interval
        subaggregate new data points, and update X
        updateACF(X)
        opt = checkLastWindow(X, opt)
        findWindow(X, opt)
    end while
end function
```

is, sliding window aggregates such as SMA can be computed more efficiently by sub-aggregating the incoming data into disjoint segments (i.e., *panes*) that are sizes of greatest common divisor of window and slide size [201]. We can perform similar pixel-aware preaggregations for data streams using panes.

ASAP maintains a linked list of all subaggregations in the window and, when prompted, re-executes the search routine from the previous section. Instead of recomputing the smoothing window from scratch, ASAP records the result of the previous rendering request and uses it as a "seed" for the new search. Specifically, since streams often exhibit similar behavior over time, the previous smoothing parameter could possibly apply to the current request. In this case, ASAP starts the new search with a known feasible window length, which enables the roughness estimation pruning procedure (isRougher in Algorithm 10) to rule out candidates automatically.

On-demand updates. A naïve strategy for updating ASAP's output is to update the plot upon arrival of each point. This is inefficient. For example, consider a data stream with a volume of one million points per second. Refreshing the plot for every data point requires updating the plot every 0.001 milliseconds. However, since humans can only perceive changes on the order of 60 events per second [163], this update rate is unnecessary. With pixel-aware preaggregation, we would refresh for each aggregated data point instead, the rate of which may still be higher than necessary. To visualize 10 minutes of data on a 27-inch iMac for example, pixel-aware preaggregation provides us aggregates data points that are 12ms apart (83Hz). As a result, we designed ASAP to only refresh at (configurable) timescales that are perceptible to humans. In our example above, a 1Hz update

speed results in a 83× reduction in the number of calls to the ASAP search routine; this reduction means we will either use less processing power and/or be able to process data at higher volumes. In Section 5.4.2, we empirically investigate the relationship between total runtime and refresh rate.

Putting it all together. Algorithm 12 shows the full streaming ASAP algorithm. ASAP aggregates the incoming data points according to the desired point-to-pixel ratio, and maintains a linked list of the aggregates. After collecting a refresh-interval-time worth of aggregates, ASAP updates data points in the current visualization, and recalculates the autocorrelation (UPDATEACF). ASAP then checks whether the window length from the last rendering request is still feasible (CHECKLASTWINDOW). If so, ASAP uses this previous window length to quickly improve the lower bound for the new search. Otherwise, ASAP starts the new search from scratch.

5.3 Evaluation: User Study

In this section, we evaluate the empirical effectiveness of ASAP's visualizations via two user studies. We demonstrate that ASAP visualizations lead to faster and more accurate identifications of anomalies in time series.

Visualization Techniques for Comparison. In each study, we compare ASAP's visualizations to a set of alternatives (cf. Section 5.5.1): *i*) the original data, *ii*) the M4 algorithm [173], *iii*) the Visvalingam-Whyatt algorithm [318], *iv*) piecewise aggregate approximation (PAA) [184] (PAA100 reduces the number of points to 100; PAA800 reduces to 800), and *v*) an "oversmoothed" plot generated by applying SMA with a window size of $\frac{1}{4}$ of the number of points. All plots are rendered using an 800 pixel resolution.

Datasets. We select five publicly-available time series described in Table 5.2 because each has known ground truth anomalies. We use this ground truth as a means of evaluating visualization quality by measuring users' ability to identify anomalous behaviors in the visualization and by assessing their preferences. Plots and text descriptions used in our user studies are available in Appendix B of the extended Technical Report [267].

5.3.1 Anomaly Identification

To assess how different smoothing algorithms affect users' ability to identify anomalies in time series visualization, we ran a large-scale user study on Amazon Mechanical Turk, in accordance with Stanford University IRB guidelines.

In this first study, we presented users with textual descriptions of each dataset and anomaly, and asked them to select one out of the five equally-sized regions in a given time series visualization where the described anomaly occurred. Users performed anomaly identification using a single, randomly chosen visualization for each dataset, and, for each identification task, we recorded the

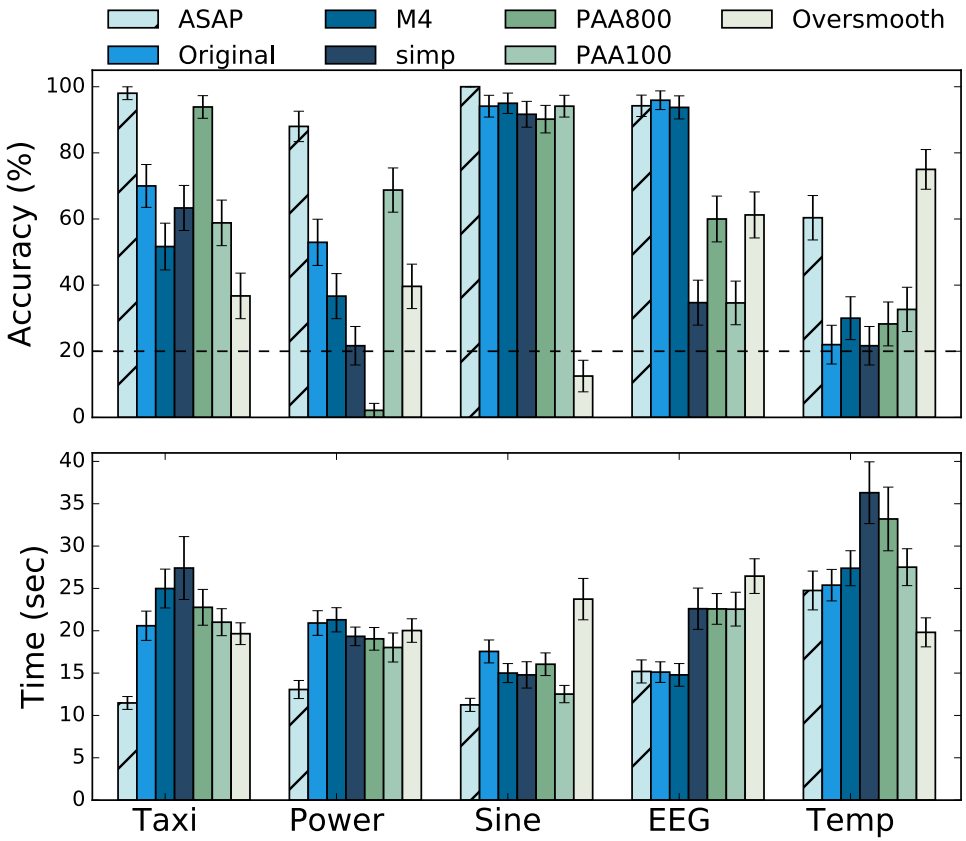


Figure 5.6: Accuracy in identifying anomalous regions and response times, with error bars indicating standard error of samples. On average, ASAP improves accuracy by 32.7% while reducing response time by 28.8% compared to other visualizations.

user's accuracy and response time. The user study involved 700 distinct Amazon Mechanical Turk workers, 406 of whom self-reported as intermediate or expert users of Excel, 324 of whom self-reported as intermediate or expert users of databases, and 288 of whom self-reported seeing time series at least once per month.

We report the accuracy and response time for the seven visualization techniques described above in Figure 5.6, where each bar in the plot represents an average of 50 users. When shown ASAP's visualizations, users were more likely to correctly identify the anomalous region, and to do so more quickly than alternatives. Specifically, users' accuracy of identifying the anomalous region increased by 21.3% when presented with ASAP's visualizations instead of the original time series, and users did so 23.9% more quickly. Compared to all other methods, users experience an average of 35.0% (max: 43.1%) increase in accuracy and 29.8% (max: 33.8%) decrease in response time with ASAP. ASAP led to most accurate results for all datasets except for the Temp dataset, in which the oversmooth

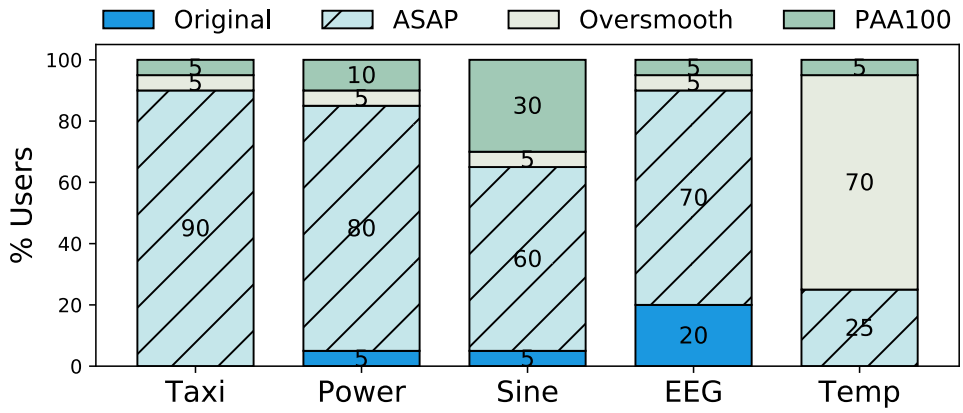


Figure 5.7: Visual preference study. Users prefer ASAP 65% of the time on average, and 59% more often than the original time series.

strategy was able to better highlight (by 14.6%) a large increasing temperature trend over several decades, corresponding to the rise of global warming [325]. However, ASAP results in 38.4% more accurate identification than the raw data for this dataset. Overall, ASAP consistently produces high-quality plots, while the quality of alternative visualization methods varies widely across datasets. We provide additional results from a sensitivity analysis of the impact of roughness and kurtosis in Section 5.3.3, where we show that ASAP also outperforms alternative configurations in average accuracy and response time.

5.3.2 Visual Preferences

In addition to the above user study, which was based on a large crowdsourced sample, we performed a targeted user study with 20 graduate students in Computer Science. We retained the same datasets and descriptions of dataset and anomaly from the previous study, and asked users to select the *visualization* that best highlights the described anomaly in order to measure visualization preferences. In contrast with the previous study, due to smaller sample size, we presented a set of four visualizations–original, ASAP, PAA100, and oversmooth–anonymized and randomly permuted for each dataset.

Figure 5.7 presents results from this study. Across all five datasets, users preferred ASAP's visualizations as a means of visualizing anomalies in 65% of the trials (random: 25%). Specifically, for datasets Taxi (Figure B.4, Appendix [267]), EEG (Figure B.5, Appendix [267]), and Power (Figure B.7, Appendix [267]), over 70% of users preferred ASAP's presentation of the time series. For these datasets, smoothing helps remove the high-frequency fluctuations in the original dataset and therefore better highlights the known anomalies. For dataset Sine (Figure B.6, Appendix [267]), a simulated noisy sine wave with a small region where the period is halved, 60% users chose ASAP, followed by 30% choosing PAA100. In follow-up interviews, some users expressed uncertainty about

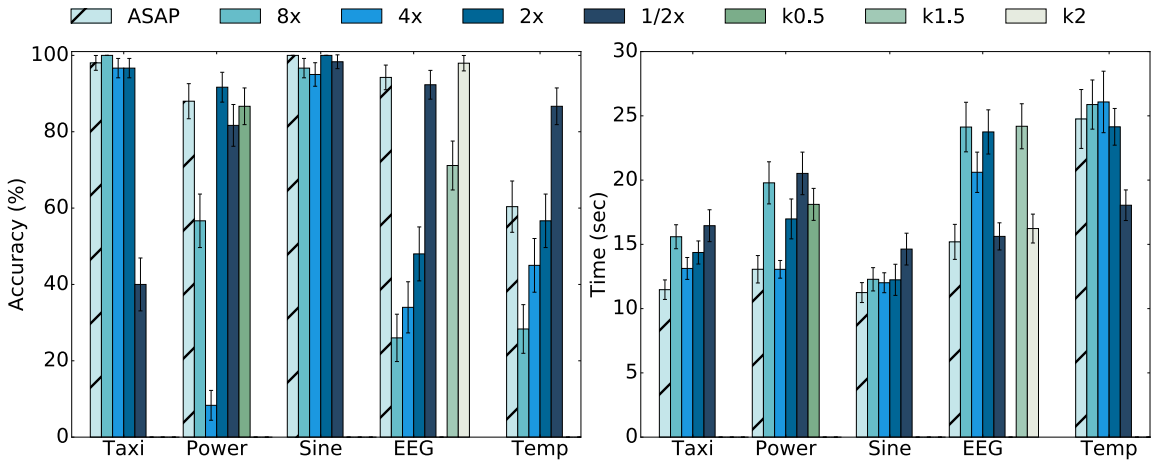


Figure 5.8: Impact of roughness and kurtosis on user's accuracy and response time for the anomaly identification user study.

this final plot: while the ASAP plot clearly highlights the anomaly, the PAA100 plot more closely resembles the description of the original signal. In the Temp dataset, 70% of users chose the over-smoothed plot, and 25% chose ASAP. For this dataset—which contains monthly temperature readings spanning over 250 years—aggressive smoothing better highlights the decade-long warming trend (Figure B.3, Appendix [267]). In addition, no user preferred the original temperature plot, further confirming that smoothing is beneficial.

In summary, these results illustrate the utility of ASAP's target metrics in producing high-quality time series visualizations that highlight anomalous behavior.

5.3.3 Sensitivity Analysis

Finally, we performed a sensitivity analysis in which we compared users' performances in the anomaly identification task under variations of the original ASAP problem formulation.

Sensitivity to Target Roughness. We varied the target roughness for each dataset and measured the impact on end user's accuracy and response time. Specifically, we used the roughness of the ASAP plot for each dataset as the reference, and generated plots that have 8 times (8x), 4 times (4x), 2 times (2x) and half (1/2x) the roughness. We report results in Figure 5.8, where the bar represents the average accuracy and response time from about 50 Amazon mechanical turk workers. While the results vary across each dataset, we observe that less smooth plots result in lower average accuracy (61.5% for 8x and 55.8% for 4x) compared to smoother plots (78.6% for 2x and 79.8% for 1/2x). Overall, ASAP achieves the highest average accuracy and the lowest response time among all configurations.

Sensitivity to Kurtosis Constraints. Similar to the roughness study, we varied the kurtosis constraint to preserving 0.5x (k0.5), 1.5x (k1.5) and 2x (k2) of kurtosis of the original time series. For three out of the five datasets, varying the kurtosis constraint didn't affect the final visualization, since the ASAP visualization already has a relatively small roughness and high kurtosis. We report additional kurtosis results for the Power and EEG dataset in Figure 5.8. Overall, we find that roughness has a larger impact on end results compared to kurtosis.

5.4 Evaluation: Performance Analysis

While the user studies illustrate the utility of the smoothed visualizations, it is critical that ASAP is able to render them quickly and over changing time series. To assess ASAP's end-to-end performance as well as the impact of each of its optimizations, we performed a series of performance benchmarks. Our goal is to demonstrate that:

- ASAP identifies high quality windows quickly (Section 5.4.1).
- ASAP's optimizations—autocorrelation, pixel-aware aggregation and on-demand update—provide complementary speedups up to seven order-of-magnitude over baseline (Section 5.4.2).

Implementation and Experimental Setup. We implemented an ASAP prototype as an explanation operator for processing output data streams in the MacroBase streaming analytics engine [29]. We report results from evaluating the prototype on a server with four Intel Xeon E5-4657L 2.4GHz CPUs containing 12 cores per CPU and 1TB of RAM (although we use considerably less RAM in processing). We exclude data loading time from our results but report all other computation time. We report results from the average of three or more trials per experiment. We use a set of 11 of datasets of varying sizes collected from a variety of application domains; Table 5.2 provides detailed descriptions of each dataset; we provide plots from each experiment in [267].

5.4.1 End-to-End Performance

To demonstrate ASAP's ability to find high-quality window sizes quickly, we evaluate ASAP's window quality and search time compared to alternative search strategies. We compare exhaustive search, grid search of varying step size (2, 10), and binary search.

First, as Table 5.2 illustrates, with a target resolution of 1200 pixels, ASAP is able to find the same smoothing parameter as the exhaustive search for all datasets by checking an average of 8.64 candidates, instead of 113.64 candidates per dataset for the exhaustive search. For the Twitter_AAPL dataset, both exhaustive search and ASAP leave the visualization unsmoothed; this time series (Figure C.4f) is smooth except for a few unusual peaks, so further smoothing would have averaged out the peaks.

Table 5.2: Dataset descriptions and batch results from ASAP and exhaustive search over pre-aggregated data for target resolution 1200 pixels. ASAP finds the same choice of smoothing parameter as optimal, exhaustive search while searching an average of 13× fewer candidates.

Dataset	Description	# points	Duration	Exhaustive	ASAP
gas_sensor [206]	Sensor measurements of a gas mixture	4,208,261	12 hours	window size: 26 #candidates: 115	window size: 26 #candidates: 7
EEG [187]	Excerpt of electrocardiogram	45,000	180 sec	window size: 22 #candidates: 119	window size: 22 #candidates: 21
Power [187]	Power consumption of a research facility	35,040	35040 sec	window size: 16 #candidates: 115	window size: 16 #candidates: 23
traffic_data [12]	Vehicle traffic observed for 4 months	32,075	4 months	window size: 84 #candidates: 120	window size: 84 #candidates: 6
machine_temp [197]	Temperature of a machine component	22,695	70 days	window size: 44 # candidates: 125	window size: 44 #candidates: 7
Twitter_AAPL [197]	A collection of Twitter mentions of Apple	15,902	2 months	window size: 1 # candidates: 120	window size: 1 #candidates: 7
ramp_traffic [206]	Car count on a freeway ramp in Los Angeles	8,640	1 month	window size: 96 # candidates: 117	window size: 96 #candidates: 5
sim_daily [197]	Simulated data with one abnormal day	4,033	2 weeks	window size: 72 # candidates: 100	window size: 72 #candidates: 5
Taxi [197]	Number of NYC taxi passengers	3,600	75 days	window size: 112 # candidates: 120	window size: 112 #candidates: 4
Temp [164]	Monthly temperature in England (1723-1970)	2,976	248 years	window size: 112 # candidates: 120	window size: 112 #candidates: 4
Sine [185]	Noisy sine wave with an anomaly that is half the usual period	800	800 sec	window size: 64 # candidates: 79	window size: 64 #candidates: 6

Second, we evaluate differences in wall-clock speed and achieved smoothness. All algorithms run on preaggregated data, so the throughput difference is only caused by the difference in search strategies; we further investigate the impact of pixel-aware preaggregation in Section 5.4.2. Figure 5.9 shows that ASAP is able to achieve up to 60× faster search time than exhaustive search over pre-aggregated series, with near-identical roughness ratio. ASAP's runtime performance scales comparably to binary search, although it lags by up to 50% due to its autocorrelation calculation. However, while ASAP produces high-quality smoothed visualizations, binary search is up to 7.5× rougher than ASAP. Grid search with step size two delivers similar-quality results as ASAP but fails to scale, while grid search with step size ten delivers the worst overall results. In summary, end-to-end, ASAP provides significant speedups over exhaustive search while retaining its quality of visualization. We provide additional runtime comparison with PAA and M4 in [267] (Appendix A.3).

5.4.2 Impact of Optimizations

In this section, we further evaluate the contribution of each of ASAP's optimizations—autocorrelation pruning, pixel-aware preaggregation, on-demand update—both individually and combined.

Pixel-aware preaggregation. We first perform a microbenchmark on the impact of pixel-aware preaggregation (Section 5.2.3) on both throughput and smoothness. Figure 5.10 shows the

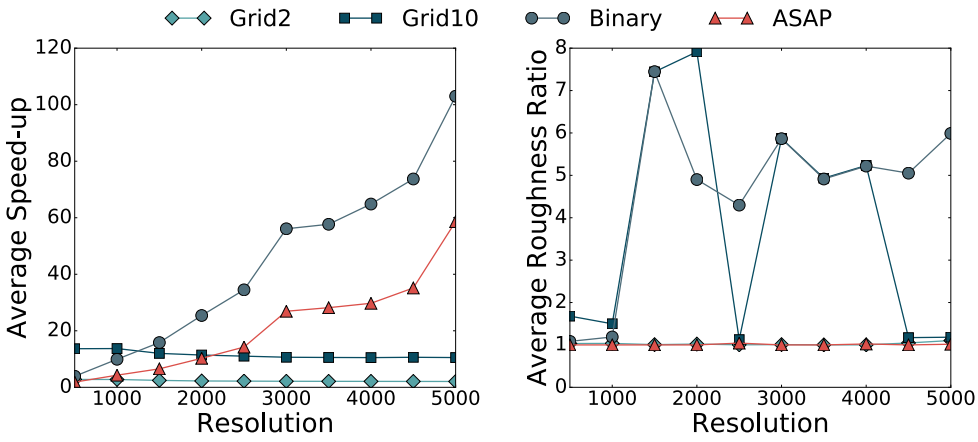


Figure 5.9: Throughput and quality of ASAP, grid search, and binary search over pre-aggregated time series according to varying target resolutions. Both plots report throughput and roughness compared to exhaustive search and report an average from the seven largest datasets in Table 5.2. ASAP exhibits similar speed-ups to binary search while retaining quality close to exhaustive search. ASAP's autocorrelation calculation incurs up to 50% overhead compared to binary search but its results are up to 7.5× smoother.

throughput and quality of ASAP and exhaustive search with and without pixel-aware preaggregation under varying target resolutions. With pixel-aware pre-aggregation, ASAP achieves roughness within 20% of exhaustive search over the raw series and sometimes outperforms exhaustive search because the initial pixel-aware preaggregation results in lower initial kurtosis. The preaggregation strategy enables a five and a 2.5 order-of-magnitude speedups over exhaustive search (Exhaustive) and ASAP on raw data (ASAPno-agg), respectively. In summary, pixel-aware preaggregation has a modest impact on result quality and massive impact on computational efficiency (i.e., sub-second versus hours to process 1M points). Should users desire exact result quality, they can still choose to disable pixel-aware preaggregation while retaining speedups from other optimizations. We provide additional analysis of pre-aggregation and additional experimental results in Appendix C.1.2.

On-demand update. To investigate the impact of the update interval in the streaming setting (Section 5.2.4), we vary ASAP's refresh rate and report throughput under each setting. The log-log plot (Figure 5.11) shows a linear relationship between the refresh interval and throughput. This is expected because updating the plot twice as often means that it would take twice as long to process the same number of points. For fast-moving streams, this strategy can save substantial computational resources.

Factor Analysis. In addition to analyzing the impact of individual optimizations, we also investigate how ASAP's three main optimizations contribute to overall performance. Figure 5.12

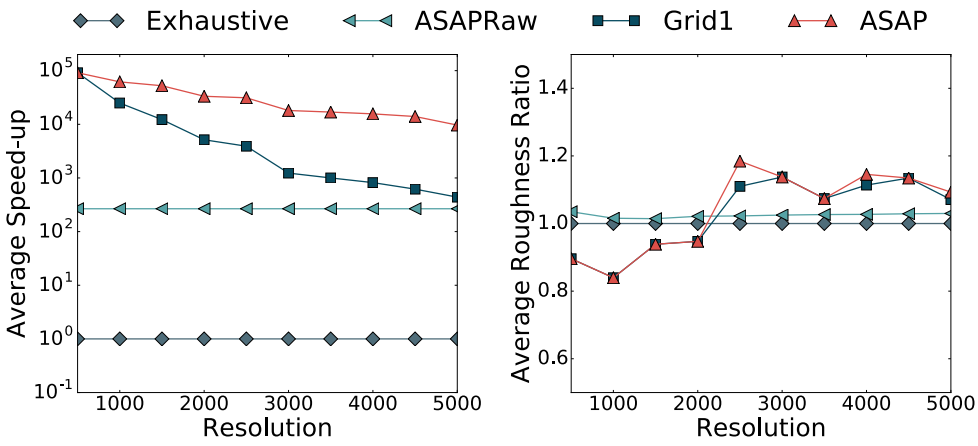


Figure 5.10: Throughput and quality of ASAP, exhaustive search on preaggregated time series over the baseline (exhaustive search over the original time series) under varying resolution. ASAP on pre-aggregated time series is up to 4 orders of magnitude faster, while retaining roughness within 1.2× of the baseline.

(left) depicts a factor analysis, where we enable each optimization cumulatively in turn. Pixel-aware aggregation provides between two and four orders of magnitude improvement depending on the target resolution. Autocorrelation provides an additional two orders of magnitude. Finally, on demand update with a daily refresh interval (updating for every 288 points in the original series versus updating for each preaggregated point) provides another two order-of-magnitude speedups. These results demonstrate that ASAP's optimizations are additive and that end-to-end, optimized streaming ASAP is approximately seven orders of magnitude faster than the baseline.

To illustrate that no one ASAP optimization is responsible for all speedups, we perform a lesion study, where we remove each optimization from ASAP while keeping the others enabled (Figure 5.12,

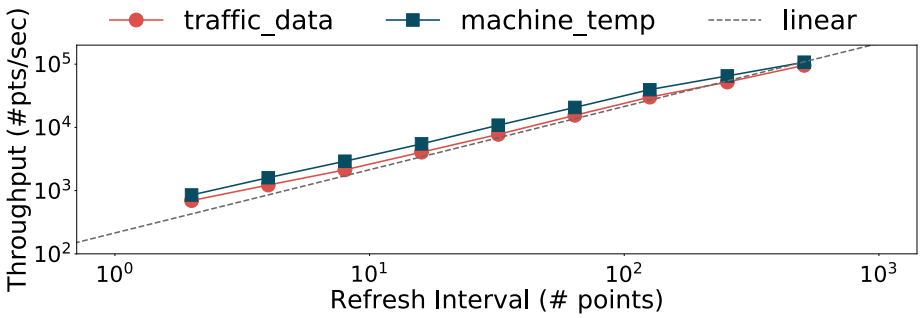


Figure 5.11: Throughput of streaming ASAP on two datasets, with varying refresh intervals (measured in the number of points) for target resolution 2000 pixels in log-log scale. The plot captures a linear relationship between throughput and refresh interval as expected.

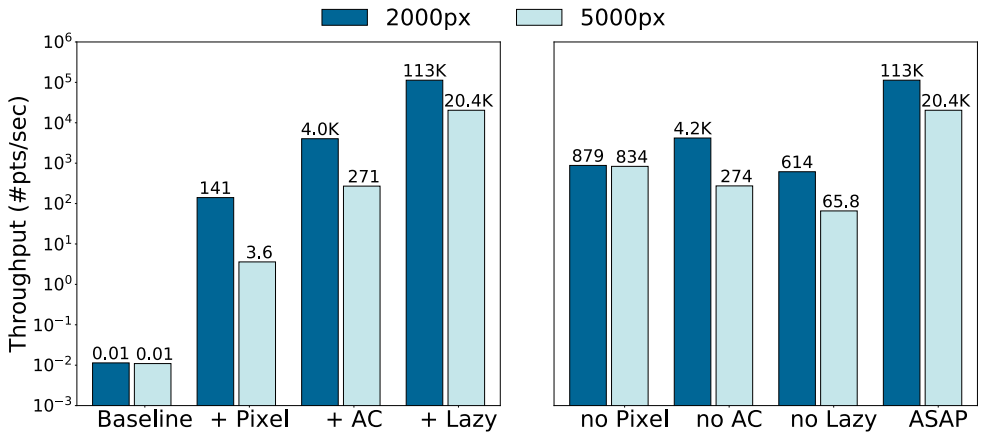


Figure 5.12: Factor analysis on machine_temp dataset under two display settings. Cumulatively enabling optimizations shows that each contributes positively to final throughput; combined, the three optimizations enable seven orders of magnitude speedup over the baseline. In addition, removing each optimization decreases the throughput by two to three orders of magnitude.

right). Removing on-demand update, pixel-aware aggregation, and autocorrelation-enabled pruning each decreases the throughput by approximately two to three orders of magnitude, in line with results from the previous experiment. Without pixel-aware preaggregation, ASAP makes no distinction between higher and lower resolution setting, so the throughput for both resolutions are near-identical. In contrast, removing the other two optimizations degrades the performance for the higher resolution setting more. Thus, each of ASAP's optimizations is necessary to achieve maximum performance.

5.5 Discussion

In this section, we discuss the related work, alternative design choices as well as real-world deployment and usage of ASAP.

5.5.1 Related Work

ASAP's design draws upon work from several domains including data visualization, signal processing and stream processing. We discuss related work in each domain below.

Time Series Visualization. Within the time series literature, which spans simplification and reduction [106,120,184,265,286,318], information retrieval [156], and data mining [10,119,207,274], visualization plays an important role in analyzing and understanding time series data [95]. There are a number of existing approaches to time series visualization [11]. Perhaps most closely related is M4 [173], which downsamples the original time series while preserving the shape—a perception-aware procedure [111]. Unlike M4 and many existing time series visualization techniques, which

focus on producing visually indistinguishable representations of the original time series (often using fewer points) by optimizing metrics such as pixel accuracy [106, 120, 265, 286, 318], ASAP visually highlights large-scale deviations in the time series by smoothing short term fluctuations.

To illustrate this difference in goals, we compared ASAP, M4 and the Visvalingam-Whyatt line simplification algorithm [318] both on pixel accuracy (Appendix B.1, [267]) and on end user accuracy of identifying anomalies (Section 5.3): ASAP is far worse at preserving pixel accuracy (up to 93% worse, average: 91.8% worse) than M4 but improves accuracy by up to 51% (average: 26.7%) for end-user anomaly identification tasks. These trends were similar for piecewise aggregate approximation [184]—which, in contrast, was *not* originally designed for visualization. Despite differing objectives, we believe that pixel-preserving visualization techniques such as M4 are complementary to ASAP: a visualization dashboard could render the original time series using M4 and overlay with ASAP to highlight long-term deviations.

Signal Processing. Noise reduction is a classic and extremely well studied problem in signal processing. Common reduction techniques include the wavelet transform [94], convolution with smoothing filters [76, 276], and non-linear filters [311]. In this work, we study a specific type of linear smoothing filter—moving average—and the problem of its automatic parameter selection. Despite its simplicity, moving average is an effective time domain filter that is optimal at reducing random noise while retaining a sharp step response (i.e., rapid rise in the data) [298]. While there are many studies on parameter selection mechanisms for various smoothing functions [221], the objective of most of the selection criteria is to minimize variants of reconstruction error to the original signal. In contrast, ASAP's quality metric is designed to visually highlight trends and large deviations, leading to a different optimization strategy. Biomedical researchers have explored ideas of selecting a moving average window size that highlights significantly deviating regions of DNA sequences [309]. ASAP adopts a similar measure for quality—namely, preserving significant deviations in time series—but is empirically more efficient than the exhaustive approach described in the study.

Stream Processing. ASAP is architected as a streaming operator and adapts stream processing techniques. As such, ASAP is compatible with and draws inspiration from the rich existing literature on architectures for combining signal processing and stream processing functionality [134, 183].

Specifically, aggregation over sliding windows has been widely recognized as a core operator over data streams. Sliding window semantics and efficient incremental maintenance techniques have been well-studied in the literature [21, 310]. ASAP adopts the sliding window aggregation model. However, instead of leaving users to select a window manually, in the parlance of machine learning, ASAP performs *hyperparameter tuning* [203] to automatically select a window that delivers smoothed plots that help improve end user's perception of long-term deviations in time series. We are unaware of any existing system—in production or in the literature—that performs this hyperparameter selection for smoothing time series plots. Thus, the primary challenge we address in this paper is efficiently and effectively performing this tuning via visualization-specific optimizations that leverage target

display resolution, the periodicity of the signal, and on-demand updates informed by the limits of human perception.

5.5.2 Usage and Reflection

ASAP is published as an open source Javascript library on Github, and has since received interest from the monitoring community. For example, ASAP has been incorporated into an open-source monitoring tool Graphite [341], the TimescaleDB Toolkit [23] and a popular JavaScript library `downsample` (more than 3000 weekly downloads) [231]. ASAP has also directly inspired an auto smoother for real-time dashboards at cloud monitoring company Datadog [26]. Below, we discuss the similarities and differences of the Datadog adaptation of ASAP in more detail.

Like ASAP, Datadog's Auto Smoother uses moving average as the smoothing function, and adopts the roughness and kurtosis metrics to quantitatively assess quality of the smoothed visualizations. However, the weighting of the two metrics is set differently in Auto Smoother to account for the level of fluctuations observed in their production monitoring data. Auto Smoother also leverages different optimization techniques. Although ASAP's design focuses on visualizing a single time series, Datadog has found that automatic smoothing also benefits dashboards with multiple time series, and that smoothing can be combined with outlier detection algorithms to highlight anomalies.

Overall, this example reinforces the usefulness of applying automatic smoothing to help focus end user attention on important behaviors in monitoring visualizations. In addition, the modifications and extensions of ASAP in production scenarios also suggest a number of interesting directions for future work, such as automatic smoothing for a group of time series.

5.6 Conclusion

In this chapter, we introduced ASAP, a new data visualization operator that automatically smooths time series to reduce noise, prioritizing user attention towards systematic deviations in visualizations. We demonstrated that ASAP's target metrics—roughness and kurtosis—produce visualizations that enable users to quickly and accurately identify deviations in time series. We also introduced three optimizations—autocorrelation-based search, pixel-aware preaggregation and on-demand update—that provide order-of-magnitude speedups over alternatives without compromising quality. Looking forward, we are interested in further improving ASAP's scalability and in further integrating ASAP with advanced analytics tasks including time series classification and alerting.

Chapter 6

FASTer: End-to-end Earthquake Detection

This chapter presents a novel application of LSH—and the associated challenges—in large-scale earthquake detection across seismic networks. Earthquake detection is particularly interesting in both its abundance of raw data and scarcity of labeled examples.

Seismic data is large. Earthquakes are monitored by seismic networks, which can contain thousands of seismometers that continuously measure ground motion and vibration. For example, Southern California contains over 500 seismic stations, each collecting continuous ground motion measurements at 100Hz. As a result, this network alone has collected over ten trillion (10^{13}) data points in the form of time series over the past decade [13].

Moreover, despite large measurement volumes, only a small fraction of earthquake events are cataloged, or confirmed and hand-labeled. As earthquake magnitudes (i.e., size) decrease, the frequency of earthquake events increases exponentially. Globally, major earthquakes (magnitude 7+) occur approximately once a month, while magnitude 2.0 and smaller earthquakes occur several thousand times a day. At low magnitudes, it is difficult to detect earthquake signals because earthquake energy approaches the noise floor and conventional seismological analyses can fail to disambiguate between signal and noise. Nevertheless, detecting these small earthquakes is important in uncovering the sources of earthquakes [107, 161], improving understanding of earthquake mechanics [248, 284], and better predicting the occurrences of future events [182].

To take advantage of the large volume of unlabeled raw measurement data, seismologists have developed an unsupervised, data-driven earthquake detection method, Fingerprint And Similarity Thresholding (FAST), based on waveform similarity [108]. Seismic sources repeatedly generate

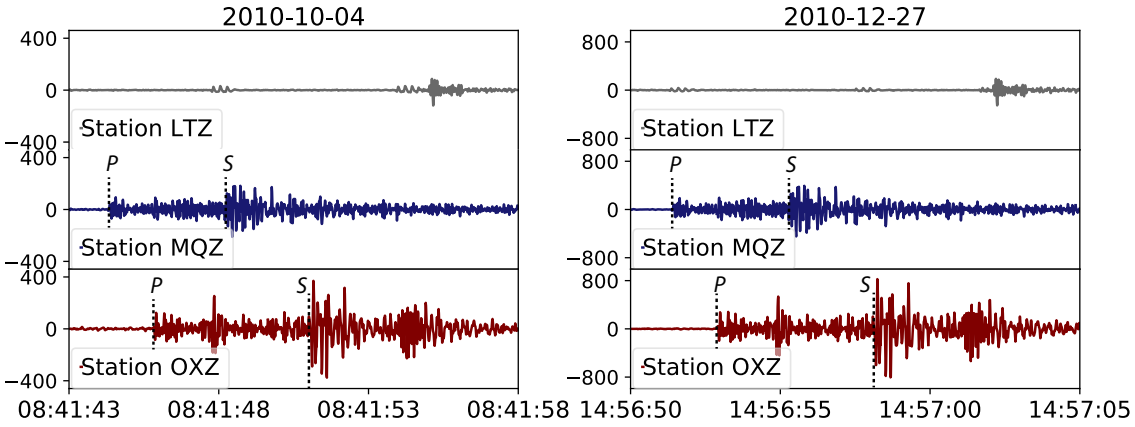


Figure 6.1: Example of near identical waveforms between occurrences of the same earthquake two months apart, observed at three seismic stations in New Zealand. The stations experience increased ground motions upon the arrivals of seismic waves (e.g., P and S waves). This paper scales LSH to over 30 billion data points and discovers 597 and 6123 new earthquakes near the Diablo Canyon nuclear power plant in California and in New Zealand, respectively.

earthquakes over the course of days, months or even years, and these earthquakes exhibit near identical waveforms when recorded at the same seismic station, regardless of the earthquake's magnitude [127, 278]. Figure 6.1 illustrates this phenomenon by depicting a pair of reoccurring earthquakes that occurred two months apart and were observed at three seismic stations in New Zealand. By applying LSH to identify similar waveforms from seismic data, seismologists were able to discover new, low-magnitude earthquakes without knowledge of prior earthquake events.

Despite early successes, the seismologists had difficulty scaling their LSH-based analysis beyond three months of time series data (7.95×10^8 data points) at a single seismic station [107]. The original implementation faced severe scalability challenges. Contrary to what theory suggests, the actual LSH runtime in FAST grew almost quadratically with the input size due to correlations in the seismic signals. In an initial performance benchmark, the similarity search took 5 CPU-days to process three months of data, and, with a 5× increase in dataset size, the LSH query time increased by 30×. In addition, station-specific repeated background noise leads to an overwhelming number of similar, non-seismic time series matches, both crippling throughput and the seismologists' ability to sift through the output, which can number in the hundreds of millions of events. Ultimately, these scalability bottlenecks prevented the seismologists from making use of the decades of data at their disposal.

This paper demonstrates how systems, algorithms, and domain expertise together can deliver substantial scalability improvements for this seismological analysis. Via algorithmic design, optimization using domain knowledge, and data engineering, we scaled the FAST workload to years of

continuous data at multiple stations. This scalability has enabled new scientific discoveries, including the detection of previously unknown earthquakes in New Zealand and near a nuclear reactor in California.

Specifically, we built a scalable end-to-end earthquake detection system FASTer[1] comprised of three main steps. First, the feature transformation step encodes the time-frequency features of the original time series into compact binary fingerprints that are more robust to small variations. To address the bottleneck caused by repeating non-seismic signals, we applied domain-specific filters based on the frequency bands and the frequency of earthquake occurrences. Second, the search step applies LSH on the binary fingerprints to identify all pairs of similar time series segments. We pinpointed high hash collision rates caused by physical correlations in the input data as a core culprit of LSH performance degradation and alleviated the impact of large buckets by increasing hash selectivity while keeping the detection threshold constant. Third, the summarization step significantly reduces the size of the detection results and confirms seismic behavior by performing spatiotemporal correlation with nearby seismic stations in the network [42]. To scale this analysis, we leveraged domain knowledge of the invariance of the time difference between a pair of earthquake events across all stations at which they were recorded.

This work makes the following contributions:

- We report on a new application of LSH in seismology as well as a complete end-to-end data science system FASTer, including non-trivial preprocessing and post-processing, that scales to a decade of continuous time series for earthquake detection.
- We present a case study for using domain knowledge to improve the accuracy and efficiency of analytics systems. We illustrate how applying seismological domain knowledge in each component of the system is critical to scalability.
- We demonstrate that our optimizations enable a cumulative two order-of-magnitude speedup in the end-to-end system FASTer. These quantitative improvements have enabled qualitative discoveries: we discovered 597 new earthquakes near the Diablo Canyon nuclear power plant in California and 6123 new earthquakes in New Zealand, allowing seismologists to determine the size and shape of nearby fault structures.

Beyond these contributions to a database audience, our solution is an open source tool, available for use by the broader scientific community. We have already held workshops for seismologists at Stanford University [114] and believe that FASTer can not only facilitate targeted seismic analyses but also contribute to the label generation for supervised methods in seismic data [250].

The rest of this chapter proceeds as follows. We review background information about earthquake detection and provide a brief overview of the end-to-end detection system and key technical challenges in Section 6.1. Sections 6.2, 6.3 and 6.4 present details as well as optimizations in the

[1] We refer to the end-to-end system as FASTer to distinguish from the earthquake detection algorithm FAST.

feature transformation, similarity search, and the summarization components of the systems. Section 6.5 presents a detailed evaluation of both the quantitative performance improvements of our optimizations as well as qualitative results of new seismic findings. Section 6.6 discusses additional related work and reflects on lessons learned.

6.1 Background and Overview

In this section, we present the necessary background knowledge in seismology to motivate our earthquake detection application (Section 6.1.1). We then give an overview of the three main components of our end-to-end earthquake detection system (Section 6.1.2).

6.1.1 Background in Seismology

With the deployment of denser and more sensitive sensor arrays, seismology is experiencing a rapid growth of high-resolution data [140]. Seismic networks with up to thousands of sensors have been recording years of continuous seismic data streams, typically at 100Hz frequencies. The rising data volume has fueled strong interest in the seismology community to develop and apply scalable data-driven algorithms that improve the monitoring and prediction of earthquake events [87, 190, 193].

In this work, we focus on the problem of detecting new, low-magnitude earthquakes from historical seismic data. Earthquakes, which are primarily caused by the rupture of geological faults, radiate energy that travels through the Earth in the form of seismic waves. Seismic waves induce ground motion that is recorded by seismometers. Modern seismometers typically include 3 components that measure simultaneous ground motion along the north-south, east-west, and vertical axes. Ground motions along each of these three axes are recorded as a separate *channel* of time series data.

Channels capture complementary signals for different seismic waves, such as the P-wave and the S-wave. The P-waves travel along the direction of propagation, like sound, while the S-waves travel perpendicular to the direction of propagation, like ocean waves. The vertical channel, therefore, better captures the up and down motions caused by the P-waves while the horizontal channels better capture the side to side motions caused by the S-waves. P-waves travel the fastest and are the first to arrive at seismic stations, followed by the slower but usually larger amplitude S-waves. Hence, the P-wave and S-wave of an earthquake typically register as two "big wiggles" on the ground motion measurements (Figure 6.1). These impulsive arrivals of seismic waves are example characteristics of earthquakes that seismologists look for in the data.

While it is easy for human eyes to identify large earthquakes on a single channel, accurately detecting small earthquakes usually requires looking at data from multiple channels or stations. These low-magnitude earthquakes pose challenges for conventional methods for detection, which we outline below. Traditional energy-based earthquake detectors such as a short-term average (STA)/long-term average (LTA) identify earthquake events by their impulsive, high signal-to-noise P-wave and S-wave

arrivals. However, these detectors are prone to high false positive and false negative rates at low magnitudes, especially with noisy backgrounds [132]. Template matching, or the waveform cross-correlation with template waveforms of known earthquakes, has proven more effective for detecting known seismic signals in noisy data [48, 279]. However, the method relies on template waveforms of prior events and is not suitable for discovering events from unknown sources.

As a result, almost all earthquakes greater than magnitude 5 are detected [110]. In comparison, an estimated 1.5 million earthquakes with magnitude between 2 and 5 are not detected by conventional means, and 1.3 million of these are between magnitude 2 and 2.9. The estimate is based on the magnitude frequency distribution of earthquakes [141]. We are interested in detecting these low-magnitude earthquakes missing from public earthquake catalogs to better understand earthquake mechanics and sources, which inform seismic hazard estimates and prediction [161, 182, 248, 284].

The earthquake detection pipeline we study in the paper is an unsupervised and data-driven approach that does not rely on supervised (i.e., labeled) examples of prior earthquake events, and is designed to complement existing, supervised detection methods. As in template matching, the method we optimize takes advantage of the high similarity between waveforms generated by reoccurring earthquakes. However, instead of relying on waveform templates from only known events, the pipeline leverages the recurring nature of seismic activities to detect similar waveforms in time and across stations. To do so, the pipeline performs an all-pair time series similarity search, treating each segment of the input waveform data as a "template" for potential earthquakes. This pipeline will not detect an earthquake that occurs only once and is not similar enough to any other earthquakes in the input data. Therefore, to improve detection recall, it is critical to be able to scale the analysis to input data with a longer duration (e.g., years instead of weeks or months).

6.1.2 System Overview

Next, we provide an overview of the three main steps of our end-to-end detection system FASTer. We elaborate on each step—and our associated optimizations—in later sections, referenced inline.

The input of the detection system consists of continuous ground motion measurements in the form of time series, collected from multiple stations in the seismic network. The output is a list of potential earthquakes, specified in the form of timestamps when the seismic wave arrives at each station. From there, seismologists can compare with public earthquake catalogs to identify new events, and visually inspect the measurements to confirm seismic findings.

Figure 6.2 illustrates the three major components of the end-to-end detection system: feature transformation, similarity search, and result summarization. For each input time series, or continuous ground motion measurements from a seismic channel, the algorithm slices the input into short windows of overlapping time series segments and encodes time-frequency features of each window into a binary fingerprint; the similarity of the fingerprints resembles that of the original waveforms (Section 6.2). The algorithm then performs an all pairs similarity search via LSH on the binary

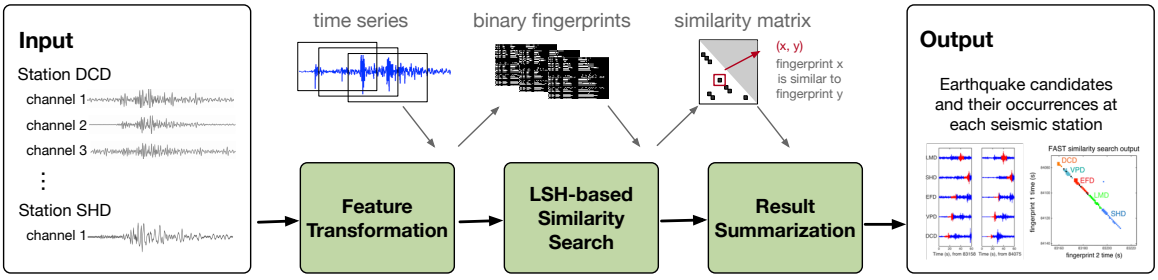


Figure 6.2: The three main components in the end-to-end earthquake detection system FASTer: fingerprinting transforms time series into binary vectors (Section 6.2); similarity search identifies pairs of similar binary vectors (Section 6.3); summarization aggregates and reduces false positives in results (Section 6.4).

fingerprints and identifies pairs of highly similar fingerprints (Section 6.3). Finally, like a traditional associator that maps earthquake detections at each station to a consistent seismic source, in the spatiotemporal alignment stage, the algorithm combines, filters and clusters the outputs from all seismic channels to generate a list of candidate earthquake detections with high confidence (Section 6.4).

A naïve implementation of the system imposes several scalability challenges. For example, we observed LSH performance degradation in our application caused by the non-uniformity and correlation in the binary fingerprints; the correlations induce undesired LSH hash collisions, which significantly increase the number of lookups per similarity search query (Section 6.3.3). In addition, the similarity search does not distinguish seismic from non-seismic signals. In the presence of repeating background signals, similar noise waveforms could outnumber similar earthquake waveforms, leading to more than an order of magnitude slow down in runtime and increase in output size (Section 6.3.5). As the input time series and the output of the similarity search becomes larger, the system must adapt to data sizes that are too large to fit into main memory (Section 6.3.4, 6.4.2).

In this work, we focus on single-machine, main-memory execution on commodity servers with multicore processors. We parallelize the system within a given server but otherwise do not distribute the computation to multiple servers. In principle, the parallelization efforts extend to distributed execution. However, given the poor quadratic scalability of the unoptimized system, distribution alone would not have been a viable option for the scalability challenges. As a result of the optimizations described in this paper, we are able to analyze a decade of data on a single node without requiring distribution. We view distributed execution as a valuable extension for future work.

In the remaining sections of this paper, we describe the design decisions as well as performance optimizations for each system component. Most of our optimizations focus on the all pairs similarity search, where the initial implementation exhibited near quadratic growth in runtime with the input size. We show in the evaluation that these optimizations enable speedups of more than two orders of magnitude in the end-to-end system.

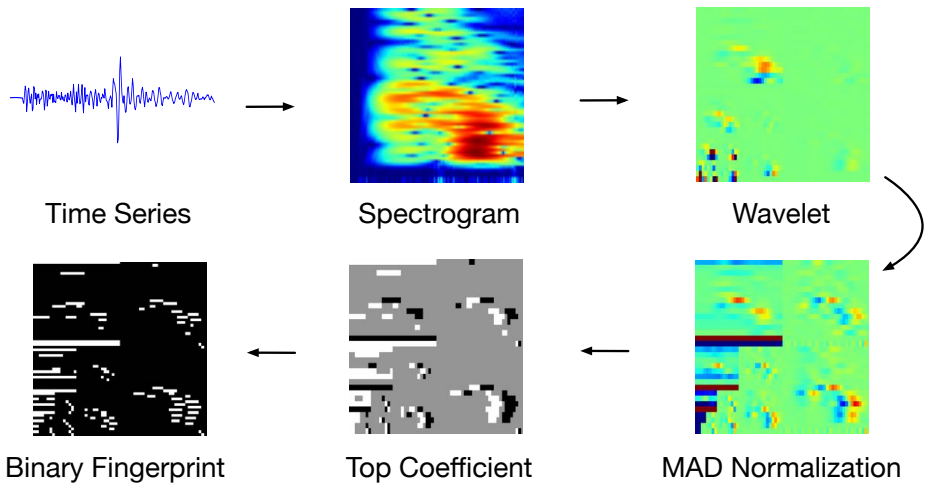


Figure 6.3: The fingerprinting algorithm encodes time-frequency features of the original time series into binary vectors.

6.2 Step One: Feature Transformation

In this section, we describe the feature transformation step that encodes time-frequency features of the input time series into compact binary vectors for similarity search. We begin with an overview of the fingerprinting algorithm [41] and the benefits of using fingerprints in place of the time series (Section 6.2.1). We then describe a new optimization that parallelizes and accelerates the fingerprinting generation via sampling (Section 6.2.2).

6.2.1 Fingerprint Overview

Inspired by the success of fingerprinting techniques for indexing audio snippets [41], the feature transformation step transforms continuous time series data into compact binary vectors (fingerprints) for similarity search. Each fingerprint encodes representative time-frequency features of the time series. The Jaccard similarity of two fingerprints, defined as the size of the intersection of the non-zero entries divided by the size of the union, preserves the waveform similarity of the corresponding time series segments. Compared to directly computing similarity on the time series, fingerprinting introduces frequency-domain features into the detection and provides additional robustness against translation and small variations [41].

Figure 6.3 illustrates the individual steps of fingerprinting:

1. **Spectrogram** Compute the spectrogram, a time-frequency representation, of the time series. Slice the spectrogram into short overlapping segments using a sliding window and smooth by downsampling each segment into a spectral image.

2. **Wavelet Transform** Compute two-dimensional discrete Haar wavelet transform on each spectral image. The wavelet coefficients serve as a lossy compression of the spectral images.
3. **Normalization** Normalize each wavelet coefficient by its median and the median absolute deviation (MAD) on the full, background dominated dataset.
4. **Top coefficient** Extract the top K most anomalous wavelet coefficients, or the largest coefficients after MAD normalization, from each spectral image. By selecting the most anomalous coefficients, we focus only on coefficients that are most distinct from coefficients that characterize noise, which empirically leads to better detection results.
5. **Binarize** Binarize the signs and positions of the top wavelet coefficients. We encode the sign of each normalized coefficient using 2 bits: $-1 \rightarrow 01$, $0 \rightarrow 00$, $1 \rightarrow 10$.

6.2.2 Optimization: MAD via sampling

The feature transformation is implemented via scientific modules such as `scipy`, `numpy` and `PyWavelets` in Python. While its runtime grows linearly with input size, fingerprinting ten years of time series data can take several days on a single core.

In particular, the normalization step in the fingerprinting algorithm remains a bottleneck for parallelization. In the unoptimized procedure, normalizing the wavelet coefficients requires two full passes over the data. The first pass calculates the median and the MAD[2] for each wavelet coefficient over the whole population, and the second pass normalizes the wavelet representation of each fingerprint accordingly. Therefore, given the median and MAD for each wavelet coefficient, the input time series can be partitioned and normalized in parallel.

We accelerate the computation by approximating the true median and MAD using a small random sample of the input data. The confidence interval for MAD with a sample size of n shrinks with $n^{1/2}$ [285]. We further investigate the trade-off between speed and accuracy under different sampling rates in the evaluation (Section 6.5.3). We empirically find that, on one month of input time series data, sampling provides an order of magnitude speedup with almost no loss in accuracy. For input time series of longer duration, sampling 1% or less of the input can suffice.

6.3 Step Two: LSH-based Similarity Search

In this section, we present the time series similar search step based on LSH. We start with a description of the algorithm and the baseline implementation (Section 6.3.1), upon which we build the optimizations. Our contributions include: an optimized hash signature generation procedure (Section 6.3.2), an empirical analysis of the impact of hash collisions and LSH parameters on query

[2]For $X = \{x_1, x_2, ..., x_n\}$, the MAD is defined as the median of the absolute deviations from the median: $MAD = median(|x_i - median(X)|)$

performance (Section 6.3.3), partition and parallelization of LSH that reduce the runtime and memory usage (Section 6.3.4), and finally, two domain-specific filters that improve both the performance and detection quality of the search (Section 6.3.5).

6.3.1 Similarity Search Overview

Reoccurring earthquakes originated from nearby seismic sources appear as near-identical waveforms at the same seismic station. Given continuous ground motion measurements collected from a seismic station, our system identifies similar time series segments from the input as candidates for such reoccurring earthquake events.

Concretely, we perform an approximate similarity search via MinHash LSH on the binary fingerprints to identify all pairs of fingerprints whose Jaccard similarity exceeds a predefined threshold [50]. MinHash LSH performs a random projection of high-dimensional data into lower dimensional space, hashing similar items to the same hash table "bucket" with high probability (Figure 6.4). Instead of performing a naïve pairwise comparisons between all fingerprints, LSH limits the comparisons to fingerprints sharing the same hash bucket, significantly reducing the computation. The ratio of the average number of comparisons per query to the size of the dataset, or *selectivity*, is a machine-independent proxy for query efficiency [105].

Hash signature generation. The MinHash of a fingerprint is the first non-zero element of the fingerprint under a given random permutation of its elements. The permutation is defined by a hash function mapping fingerprint elements to random indices. Let p denote the collision probability of a hash signature generated with a single hash function. By increasing the number of hash functions k, the collision probability of the hash signature decreases to p^k [200].

Hash table construction. Each hash table stores an independent mapping of fingerprints to hash buckets. The tables are initialized by mapping hash signatures to a list of fingerprints that share the same signature. Empirically, we find that using $t = 100$ hash tables suffices for our application, and there is little gain in further increasing the number of hash tables.

Search. The search queries the hash tables for each fingerprint's near neighbor candidates, or other fingerprints that share the query fingerprint's hash buckets. We keep track of the number of times the query fingerprint and candidates have matching hash signatures in the hash tables, and output candidates with matches above a predefined threshold. The number of matches is also used as a proxy for the confidence of the similarity in the final step of the system.

6.3.2 Optimization: Hash signature generation

In this subsection, we present both memory access patterns and algorithmic improvements to speed up the generation of hash signatures. We show that, together, the optimizations lead to an over 3× improvement in hash generation time (Section 6.5.1).

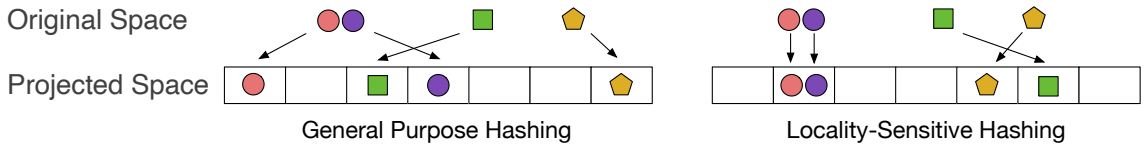


Figure 6.4: Locality-sensitive hashing hashes similar items in the high-dimensional space to the same hash "bucket" in the low-dimensional space with high probability.

Similar to observations made for SimHash (a different hash family for angular distances) [308], a naïve implementation of the MinHash generation can suffer from poor memory locality due to the sparsity of input data. SimHash functions are evaluated as a dot product between the input and hash mapping vectors, while MinHash functions are evaluated as a minimum of hash mappings corresponding to non-zero elements of the input. For sparse input, both functions access scattered, non-contiguous elements in the hash mapping vector, causing an increase in cache misses. We improve the memory access pattern by blocking the access to the hash mappings. We use dimensions of the fingerprint, rather than hash functions, as the main loop for each fingerprint. As a result, the lookups for each non-zero element in the fingerprint are blocked into rows in the hash mapping array. For our application, this loop order has the additional advantage of exploiting the high overlap (e.g. over 60% in one example) between neighboring fingerprints. The overlap means that previously accessed elements in hash mappings are likely to get reused while in cache, further improving the memory locality.

In addition, we speed up the hash signature generation by replacing MinHash with Min-Max hash. MinHash only keeps the minimum value for each hash mapping, while Min-Max hashkeeps both the min and the max. Therefore, to generate hash signatures with similar collision probability, Min-Max hash reduces the number of required hash functions to half. Previous work showed the Min-Max hash is an unbiased estimator of pairwise Jaccard similarity, and achieves similar and sometimes smaller mean squared error (MSE) in estimating pairwise Jaccard similarity in practice [170]. We include pseudocode for the optimized hash signature calculation in Appendix D of [270].

6.3.3 Optimization: Alleviating hash collisions

Perhaps surprisingly, our initial LSH implementation demonstrated poor scaling with the input size: with a 5× increase in input, the runtime increases by 30×. We analyze the cause of LSH performance degradation and the performance implications of core LSH parameters in our application below.

Cause of hash collisions. Poor distribution of hash signatures can lead to large LSH hash buckets or high query *selectivity*, significantly degrading the performance of the similarity search [34, 180]. For example, in the extreme case when all fingerprints are hashed into a single bucket, the *selectivity* equals 1 and the LSH performance is equivalent to that of the exhaustive $O(n^2)$ search.

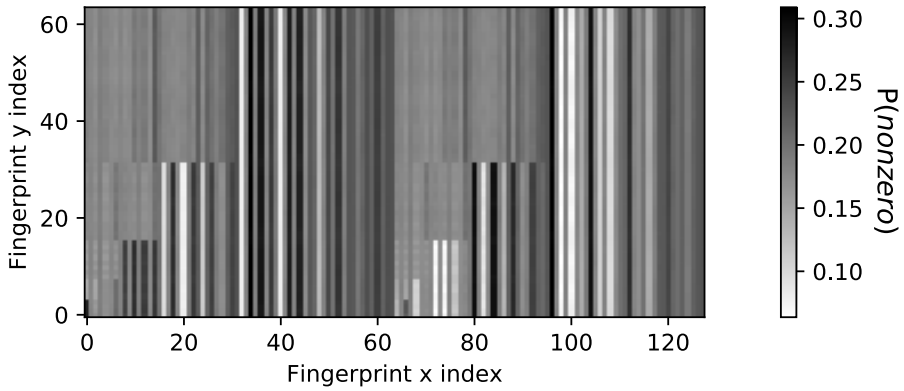


Figure 6.5: Probability that each element in the fingerprint is equal to 1, averaged over 15.7M fingerprints, each of dimension 8192, generated from a year of time series data. The heatmap shows that some elements of the fingerprint are much more likely to be non-zero compared to others.

Our input fingerprints encode physical properties of the waveform data. As a result, the probability that each element in the fingerprint is non-zero is highly non-uniform (Figure 6.5). Moreover, fingerprint elements are not necessarily independent, meaning that certain fingerprint elements are likely to co-occur: given an element a_i is non-zero, the element a_j has a much higher probability of being non-zero ($\mathbb{P}[a_i = 1, a_j = 1] > \mathbb{P}[a_i = 1] \times \mathbb{P}[a_j = 1]$).

This correlation has a direct impact on the collision probability of MinHash signatures. For example, if a hash signature contains k independent MinHash of a fingerprint and two of the non-zero elements responsible for the MinHash are dependent, then the signature has effectively similar collision probability as the signature with only $k-1$ MinHash . In other words, more fingerprints are likely to be hashed to the same bucket under this signature. For fingerprints shown in Figure 6.5, the largest 0.1% of the hash buckets contain an average of 32.9% of the total fingerprints for hash tables constructed with 6 hash functions.

Performance impact of LSH parameters. The precision and recall of the LSH can be tuned via two key parameters: the number of hash functions k and the number of hash table matches m. Intuitively, using k hash functions is equivalent to requiring two fingerprints agree at k randomly selected non-zero positions. Therefore, the larger the number of hash functions, the lower the probability of collision. To improve recall, we increase the number of independent permutations to make sure that similar fingerprints can land in the same hash bucket with high probability.

Formally, given two fingerprints with Jaccard similarity s, the probability that the fingerprints are hashed to the same bucket at least m times out of $t = 100$ hash tables with k hash functions is:

$$\mathbb{P}[s] = 1 - \sum_{i=0}^{m-1} [\binom{t}{i} (1 - s^k)^{t-i} (s^k)^i].$$

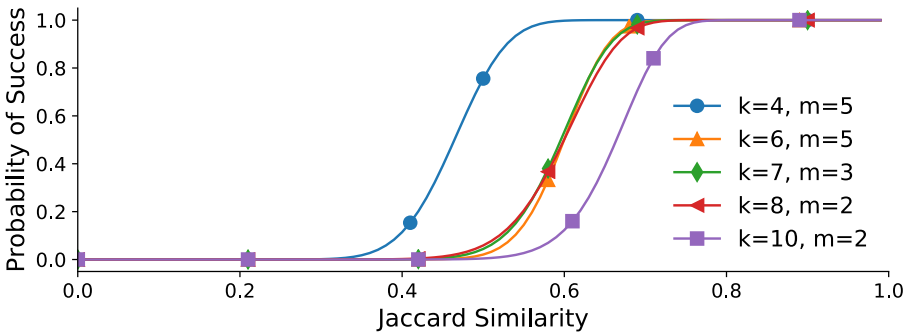


Figure 6.6: Theoretical probability of a successful search versus Jaccard similarity between fingerprints (k: number of hash functions, m: number of matches). Different LSH parameter settings can have near identical detection probability with vastly different runtime.

The probability of detection success as a function of Jaccard similarity has the form of an S-curve (Figure 6.6). The S-curve shifts to the right with the increase in the number of hash functions k or the number of matches m, increasing the Jaccard similarity threshold for LSH. Figure 6.6 illustrates a near-identical probability of success curve under different parameter settings.

Due to the presence of correlations in the input data, LSH parameters with the same theoretically success probability can have vastly different runtime in practice. Specifically, as the number of hash functions increases, the expected average size of hash buckets decreases, which can lead to an order of magnitude speed up in the similarity search for seismic data in practice. However, to keep the success probability curve constant with increased hash functions, the number of matches needs to be lowered, which increases the probability of spurious matches. These spurious matches can be suppressed by scaling up the number of total hash tables, at the cost of larger memory usage. We further investigate the performance impact of LSH parameters in the evaluation.

6.3.4 Optimization: Partitioning

In this subsection, we describe the partition and parallelization of the LSH that further reduces its runtime and memory footprint.

Partition. Using a 1-second lag for adjacent fingerprints results in around 300M total fingerprints for 10 years of time series data. Given a hash signature of 64 bits and 100 total hash tables, the total size of hash signatures is approximately 250 GB. To avoid expensive disk I/O, we also want to keep all hash tables in memory for lookups. Taken together, this requires several hundred gigabytes of memory, which can exceed available main memory.

To scale to larger input data on a single node with the existing LSH implementation, we perform similarity search in partitions. We evenly partition the fingerprints and populate the hash tables with one partition at a time, while still keeping the lookup table of fingerprints to hash signatures in

memory. During query time, we output matches between fingerprints in the current partition (or in the hash tables) with all other fingerprints and subsequently repeat this process for each partition. The partitioned search yields identical results to the original search, with the benefit that only a subset of the fingerprints are stored in the hash tables in memory. We can partition the lookup table of hash signatures similarly to further reduce memory. We illustrate the performance and memory trade-offs under different numbers of partitions in Section 6.5.3.

The idea of populating the hash table with a subset of the input could also be favorable for performing a small number of nearest neighbor queries on a large dataset, e.g., a thousand queries on a million items. There are two ways to execute the queries. We can hash the full dataset and then perform a thousand queries to retrieve near neighbor candidates in each query item's hash buckets; alternatively, we can hash only the query items and for every other item in the dataset, check whether it is mapped to an existing bucket in the table. While the two methods yield identical query results, the latter could be 8.6× faster since the cost of initializing the hash table dominates that of the search.

It is possible to further improve LSH performance and memory usage with the more space efficient variants such as multi-probe LSH [211]. However, given that the alignment step uses the number of hash buckets shared between fingerprints as a proxy for similarity, and that switching to a multi-probe implementation would alter this similarity measure, we preserve the original LSH implementation for backwards compatibility with FAST. We compare against alternative LSH implementations and demonstrate the potential benefits of adopting multi-probe LSH in the evaluation (Section 6.5.4).

Parallelization. Once the hash mappings are generated, we can easily partition the input fingerprints and generate the hash signatures in parallel. Similarly, the query procedure can be parallelized by running nearest neighbor queries for different fingerprints and outputting results to files in parallel. We show in Section 6.5.3 that the total hash signature generation time and similarity search time reduces almost linearly with the number of processes.

6.3.5 Optimization: Domain-specific filters

Like many other sensor measurements, seismometer readings can be noisy. In this subsection, we address a practical challenge of the detection system, where similar non-seismic signals dominate seismic findings in runtime and detection results. We show that by leveraging domain knowledge, we can greatly increase both the efficiency and the quality of the detection.

Filtering irrelevant frequencies. Some input time series contain station-specific narrow-band noise that repeats over time. Patterns of the repeating noise are captured in the fingerprints and are identified as near neighbors, or earthquake candidates in the similarity search.

To address this problem, we apply a bandpass filter to exclude frequency bands that show high

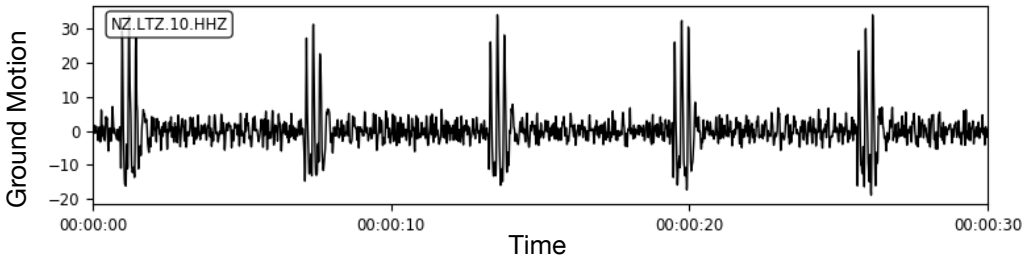


Figure 6.7: The short, three-spike pattern is an example of similar and repeating background signals not due to seismic activity. These repeating noise patterns cause scalability challenges for LSH.

average amplitudes and repeating patterns while containing low seismic activities. The bandpass filter is selected manually by examining short spectrogram samples, typically an hour long, of the input time series, based on seismological knowledge. Typical bandpass filter ranges span from 2 to 20Hz. Prior work [41, 42, 107, 108] proposes the idea of filtering irrelevant frequencies, but only on input time series. We extend the filter to the fingerprinting algorithm and cutoff spectrograms at the corner of the bandpass filter, which empirically improves detection performance. We perform a quantitative evaluation of the impact of bandpass filters on both the runtime and result quality (Section 6.5.2).

Removing correlated noise. Repeating non-seismic signals can also occur in frequency bands containing rich earthquake signals. Figure 6.7 shows an example of strong repeating background signals from a New Zealand seismic station. A large cluster of repeating signals with high pairwise similarity could produce nearest neighbor matches that dominate the search, leading to a 10× increase in runtime and an over 100× increase in output size compared to results from similar stations. This poses problems both for computational scalability and for seismological interpretability.

We develop an occurrence filter for the similarity search by exploiting the rarity of the earthquake signals. Specifically, if a specific fingerprint is generating too many nearest neighbor matches in a short duration of time, we can be fairly confident that it is not an earthquake signal. This observation holds in general except for special scenarios such as volcanic earthquakes [39].

During similarity search, we dynamically generate a list of fingerprints to exclude from future searches. If the number of near neighbor candidates a fingerprint generates is larger than a predefined percentage of the total fingerprints, we exclude this fingerprint and its neighbors from future similarity search. The filter can be applied on top of the partitioned search to capture repeating noise over a short duration of time. In this case, the filtering threshold is defined as the percentage of fingerprints in the current partition, rather than in the whole dataset. On the example dataset above, this approach filtered out around 30% of the total fingerprints with no false positives. We evaluate the effect of the occurrence filter under different filtering thresholds in Section 6.5.2.

6.4 Step Three: Result Summarization

The LSH-based similar search outputs pairs of similar fingerprints (or waveforms) from the input, without knowing whether or not the pairs correspond to actual earthquake events. In this section, we show that by incorporating domain knowledge, we are able to significantly reduce the size of the output and prioritize seismic findings in the similarity search results. We briefly summarize the aggregation and filtering techniques on the level of seismic channels, seismic stations and seismic networks introduced in a recent paper in seismology [42] (Section 6.4.1). We then describe the implementation challenges and our out-of-core adaptations enabling the algorithm to scale to large output volumes (Section 6.4.2).

6.4.1 Summarization Overview

The similarity search computes a sparse similarity matrix $\mathcal{M}$, where the non-zero entry $\mathcal{M}[i, j]$ represents the similarity of fingerprints i and j. In order to identify weak events in low signal-to-noise ratio settings, seismologists set lenient detection thresholds for the similarity search, resulting in large outputs in practice. For example, one year of input time series data can easily generate 100G of output, or more than 5 billion pairs of similar fingerprints. Since it is infeasible for seismologists to inspect all results manually, we need to automatically filter and align the similar fingerprint pairs into a list of potential earthquakes with high confidence. Based on algorithms proposed in a recent work in seismology [42], we seek to reduce similarity search results at the level of seismic channels, stations and across a seismic network. Figure 6.8 gives an overview of the summarization procedure.

Channel Level. Seismic channels at the same station experience ground movements at the same time. Therefore, we can directly merge detection results from each channel of the station by summing the corresponding similarity matrix. Given that earthquake-triggered fingerprint matches tend to register at multiple channels whereas matches induced by local noise might only appear on one channel, we can prune detections by imposing a slightly higher similarity threshold on the combined similarity matrix. This is to make sure that we include either matches with high similarity, or weaker matches registered at more than one channel.

Station Level. Given a combined similarity matrix for each seismic station, domain scientists have found that earthquake events can be characterized by thin diagonal shaped clusters in the matrix, which corresponds to a group of similar fingerprint pairs separated by a constant offset [42]. The constant offset represents the time difference, or the inter-event time, between a pair of reoccurring earthquake events. One pair of reoccurring earthquake events can generate multiple fingerprint matches in the similarity matrix, since event waveforms are longer than a fingerprint time window. We exclude "self-matches" generated from adjacent/overlapping fingerprints that are not attributable to reoccurring earthquakes. After grouping similar fingerprint pairs into clusters of thin diagonals, we reduce each cluster to a few summary statistics, such as the bounding box of

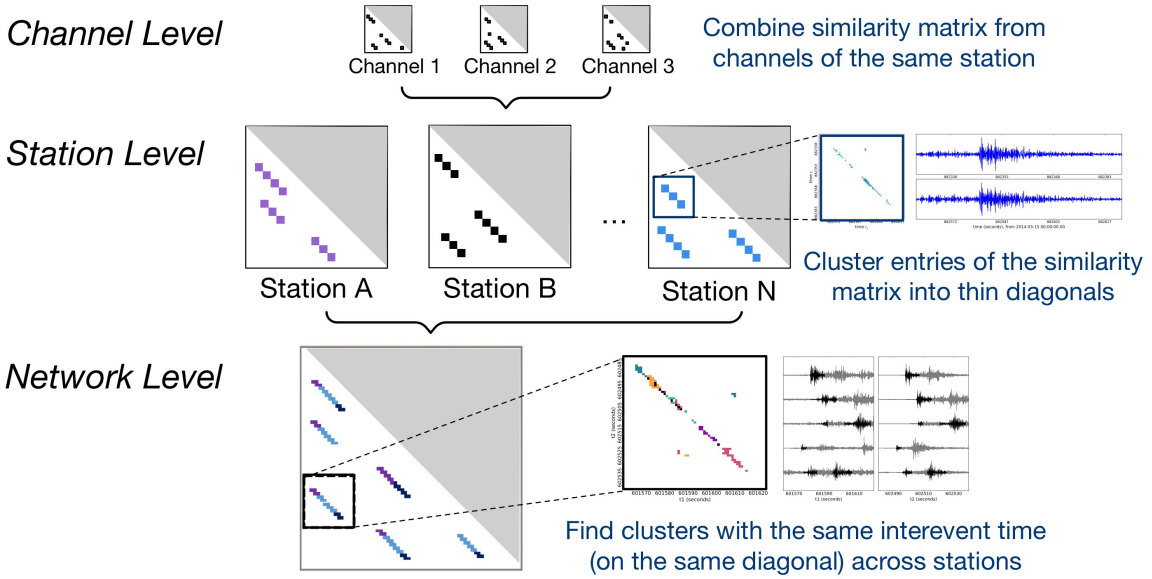


Figure 6.8: The summarization procedure combines similarity search outputs from all channels in the same station (Channel Level), groups similar fingerprint matches generated from the same pair of reoccurring earthquakes (Station Level), and checks across seismic stations to reduce false positives in the final detection list (Network Level).

the diagonal, the total number of similar pairs in the bounding box, and the sum of their similarity. Compared to storing every similar fingerprint pair, the clusters and summary statistics significantly reduce the size of the output.

Network Level. Earthquake signals also show strong temporal correlation across the seismic network, which we exploit to further suppress non-earthquake matches. Since an earthquake's travel time is only a function of its distance from the source but not of the magnitude, reoccurring earthquakes generated from the same source take a fixed travel time from the source to the seismic stations on each occurrence. Assume that an earthquake originated from source X takes δt_A and δt_B to travel to seismic stations A and B and that the source generates two earthquakes at time t_1 and t_2 (Figure 6.9). Station A experiences the arrivals of the two earthquakes at time $t_1 + \delta t_A$ and $t_2 + \delta t_A$, while station B experiences the arrivals at $t_1 + \delta t_B$ and $t_2 + \delta t_B$. The inter-event time Δt of these two earthquake events is independent of the location of the stations:

$$\Delta t = (t_2 + \delta t_A) - (t_1 + \delta t_A) = (t_2 + \delta t_B) - (t_1 + \delta t_B) = t_2 - t_1.$$

This means that in practice, diagonals with the same offset Δt and close starting times at multiple stations can be attributed to the same earthquake event. We require a pair of earthquake events to be observed at more than a user-specified number of stations in order to be considered as a detection.

On a run with 7 to 10 years of time series data from 11 seismic stations (27 channels), the

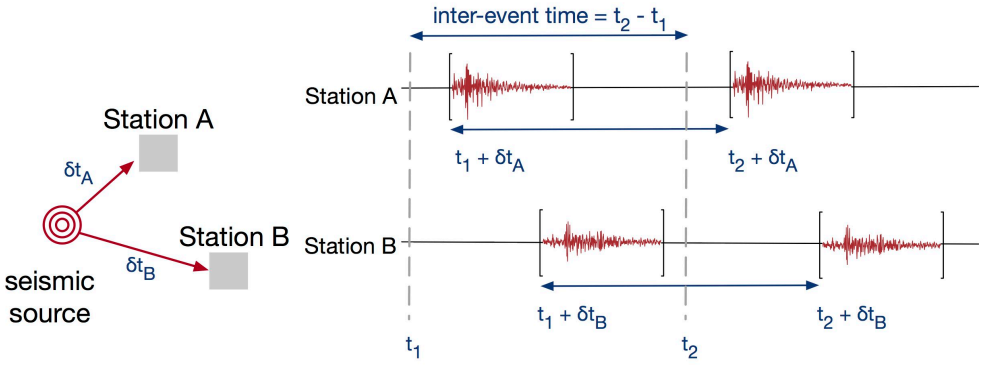


Figure 6.9: Earthquakes from the same seismic sources has a fixed travel time to each seismic station (e.g. δt_A, δt_B in the figure). The inter-event time between two occurrences of the same earthquake is invariant across seismic stations.

postprocessing procedure effectively reduced the output from more than 2 Terabytes of similar fingerprint pairs to around 30K timestamps of potential earthquakes.

6.4.2 Implementation and Optimization

The volume of similarity search output poses serious challenges for the summarization procedure, as we often need to process results larger than the main memory of a single node. In this subsection, we describe our implementation and the new out-of-core adaptations of the algorithm that enable the scaling to large output volumes.

Similarity search output format. The similarity search produces outputs that are in the form of triplets. A triplet $(dt, idx1, sim)$ is a non-zero entry in the similarity matrix, which represents that fingerprint $idx1$ and $(idx1 + dt)$ are hashed into the same bucket sim times (out of t independent trials). We use sim as an approximation of the similarity between the two fingerprints.

Channel. First, given outputs of similar fingerprint pairs (or the non-zero entries of the similarity matrix) from different channels at the same station, we want to compute the combined similarity matrix with only entries above a predefined threshold.

Naïvely, we could update a shared hashmap of the non-zero entries of the similarity matrix for each channel in the station. However, since the hashmap might not fit in the main memory on a single machine, we utilize the following sort-merge-reduce procedure instead:

1. In the sorting phase, we perform an external merge sort on the outputs from each channel, with dt as the primary sort key and $idx1$ as the secondary sort key. That is, we sort the similar fingerprint pairs first by the diagonal that they belong to in the similarity matrix, and within the diagonals, by the start time of the pairs.

2. In the merging phase, we perform a similar external merge sort on the already sorted outputs from each channel. This is to make sure that all matches generated by the same pair of fingerprint $idx1$ and $idx1 + dt$ at different channels can be concentrated in consecutive rows of the merged file.

3. In the reduce phase, we traverse through the merged file and combine the similarity score of consecutive rows of the file that share the same dt and $idx1$. We discard results that have combined similarity smaller than the threshold.

Station. Given a combined similarity matrix for each seismic station, represented in the form of its non-zero entries sorted by their corresponding diagonals and starting time, we want to cluster fingerprint matches generated by potential earthquake events, or cluster non-zero entries along the narrow diagonals in the matrix.

We look for sequences of detections (non-zero entries) along each diagonal dt, where the largest gap between consecutive detections is smaller than a predefined gap parameter. Empirically, permitting a gap helps ensure an earthquake's P and S wave arrivals are assigned to the same cluster. Identification of the initial clusters along each diagonal dt requires a linear pass through the similarity matrix. We then interactively merge clusters in adjacent diagonals $dt - 1$ and $dt + 1$, with the restriction that the final cluster has a relatively narrow width. We store a few summary statistics for each cluster (e.g. the cluster's bounding box, the total number of entries) as well as prune small clusters and isolated fingerprint matches, which significantly reduces the output size.

The station level clustering dominates the runtime in the summarization. In order to speed up the clustering, we partition the similarity matrix according to the diagonals, or ranges of dts of the matched fingerprints, and perform clustering in parallel on each partition. A naïve equal-sized partition of the similarity matrix could lead to missed detections if a cluster split into two partitions gets pruned in both due to the decrease in size. Instead, we look for proper points of partition in the similarity matrix where there is a small gap between neighboring occupied diagonals. Again, we take advantage of the ordered nature of similarity matrix entries. We uniformly sample entries in the similarity matrix, and for every pair of neighboring sampled entries, we only check the entries in between for partition points if the two sampled entries lie on diagonals far apart enough to be in two partitions. Empirically, a sampling rate of around 1% works well for our datasets in that most sampled entries are skipped because they are too close to be partitioned.

Network. Given groups of potential events at each station, we perform a similar summarization across the network in order to identify subsets of the events that can be attributed to the same seismic source. In principle, we could also partition and parallelize the network detection. In practice, however, we found that the summarized event information at each station is already small enough that it suffices to compute in serial.

6.5 Evaluation

In this section, we perform both quantitative evaluation on performances of the detection system, as well as qualitative analysis of the detection results. Our goal is to demonstrate that:

1. Each of our optimizations contributes meaningfully to the performance improvement; together, our optimizations enable an over 100× speed up in the end-to-end detection system.

2. Incorporating domain knowledge in the system improves both the performance and the quality of the detection.

3. The improved scalability of the system enables new scientific discoveries on two public datasets: we discovered 597 new earthquakes from a decade of seismic data near the Diablo Canyon nuclear power plant in California, as well as 6123 new earthquakes from a year of seismic data from New Zealand.

Dataset. We evaluate using two public datasets used in seismological analyses with our domain collaborators. The first dataset includes 1 year of 100Hz time series data (3.15 billion points per station) from 5 seismic stations (LTZ, MQZ, KHZ, THZ, OXZ) in New Zealand. We use the vertical channel (usually the least noisy) from each station [129]. The second dataset of interest includes 7 to 10 years of 100Hz time series data from 11 seismic stations and 27 total channels near the Diablo Canyon power plant in California [233].

Experimental Setup. We report results from evaluating the system on a server with 512GB of RAM and two 28-thread Intel Xeon E5-2690 v4 2.6GHz CPUs. Our test server has L1, L2, L3 cache sizes of 32K, 256K and 35840K. We report the runtime averages from multiple trials.

6.5.1 End-to-end Evaluation

In this subsection, we report the runtime breakdown of the baseline implementation of the system, as well as the effects of applying different optimizations.

To evaluate how our optimizations scale with data size, we evaluate the end-to-end system on 1 month and 1 year of time series data from station LTZ in the New Zealand dataset. We applied a bandpass filter of 3-20Hz on the original time series to exclude noisy low-frequency bands. For fingerprinting, we used a sliding window with length of 30 seconds and slide of 2 seconds, which results in 1.28M binary fingerprints for 1 month of time series data (15.7M for one year), each of dimension 8192; for similarity search, we use 6 hash functions, and require a detection threshold of 5 matches out of 100 hash tables. We further investigate the effect of varying these parameters in the microbenchmarks in Section 6.5.3.

Figure 6.10 shows the cumulative runtime after applying each optimization. Table 6.1 shows the detailed runtime breakdown for the evaluation on 1 year of data. Cumulatively, our optimizations

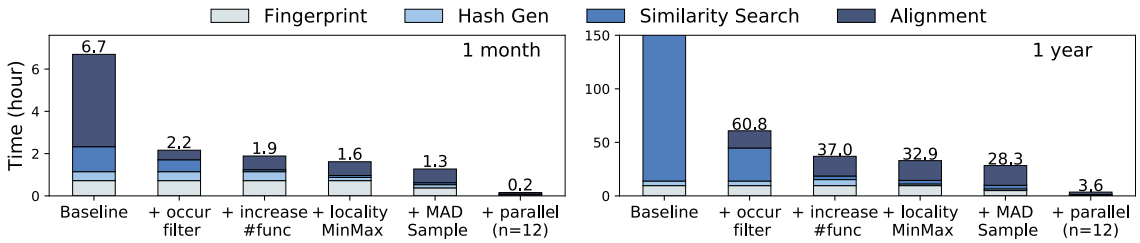


Figure 6.10: Factor analysis of processing 1 month (left) and 1 year (right) of 100Hz data from LTZ station in the New Zealand dataset. We show that each of our optimization contributes to the performance improvements, and enabled an over 100× speed up end-to-end.

Stages	**Fingerprint**	**Hash Gen**	**Search**	**Alignment**
Baseline	9.58	4.28	149	>1 mo (est.)
+ occur filter	9.58	4.28	**30.9** (-79%)	**16.02**
+ #n func	9.58	**5.63** (+32%)	**3.35** (-89%)	**18.42** (+15%)
+ locality Min-Max	9.58	**1.58** (-72%)	3.35	18.42
+ MAD sample	**4.98** (-48%)	1.58	3.35	18.42
+ parallel (n=12)	**0.54** (-89%)	**0.14** (-91%)	**0.62** (-81%)	**2.25** (-88%)

Table 6.1: Factor analysis (runtime in hours, and relative improvement) of each optimization on 1 year of data from station LTZ. Each optimization contributes meaningfully to the speedup of the system, and together, the optimizations enable an over 100× end-to-end speedup.

scale well with the size of the dataset, and enable an over 100× improvement in end-to-end processing time. We analyze each of these components in turn:

First, we apply a 1% occurrence filter (+ occur filter, Section 6.3.5) during similarity search to exclude frequent fingerprint matches generated by repeating background noise. This enables a 2-5× improvement in similarity search runtime while reducing the output size by 10-50×, reflected in the decrease in postprocessing time.

Second, we further reduce the search time by increasing the number of hash functions to 8 and lowering the detection threshold to 2 (+ increase #funcs, Section 6.3.3). While this increases the hash signature generation and output size, it enables around 10× improvement in search time for both datasets.

Third, we reduce the hash signature generation time by improving the cache locality and reducing the computation with Min-Max hash instead of MinHash (+ locality MinMax, Section 6.3.2), which leads to a 3× speedup for both datasets.

Fourth, we speed up fingerprinting by 2× by estimating MAD statistics with a 10% sample (+ MAD sample, Section 6.2.2).

Finally, we enable parallelism and run the system with 12 threads (Section 6.2.2, 6.3.4, 6.4.2). As a result, we see an almost linear decrease in runtime in each part of the system. Notably, due

to the overall lack of data dependencies in the analysis, simple parallelization can already enable significant speedups.

The improved scalability enables us to scale analytics from 3 months to over 10 years of data. We discuss qualitative detection results from both datasets in Section 6.5.6.

6.5.2 Effect of Domain-specific Optimizations

Here, we investigate the effect of applying domain-specific optimizations to the system. We demonstrate that incorporating domain knowledge could improve both performance and result quality.

Occurrence filter. We evaluate the effect of applying the occurrence filter during similarity search on the five stations from the New Zealand dataset. For this evaluation, we use a partition size of 1 month as the duration for the occurrence threshold; a >1% threshold indicates that a fingerprint matches over 1% (10K) other fingerprints in the same month. We report the total percentage of filtered fingerprints under varying thresholds in Table 6.2. We also evaluate the accuracy of the occurrence filter by comparing the timestamps of filtered fingerprints with the catalog of the arrival times of known earthquakes at each station. In Table 6.2, we report the false positive rate, or the number of filtered earthquakes over the total number of cataloged events, of the filter under varying thresholds.

The results show that as the occurrence filter becomes stronger, the percentage of filtered fingerprints and the false positive rate both increase. For seismic stations suffering from correlated noise, the occurrence filter can effectively eliminate a significant amount of fingerprints from the similarity search. For station LTZ, a >1% threshold filters out up to 30% of the total fingerprints without any false positives, which results in a 4× improvement in runtime. For other stations, the occurrence filter has little influence on the results. This is expected since these stations do not have repeating noise signals present at station LTZ (Figure 6.7). In practice, correlated noise is rather prevalent in seismic data. In the Diablo Canyon dataset for example, we applied the occurrence filter on three out of the eleven seismic stations in order for the similarity search to finish in a tractable time.

Bandpass filter. We compare similarity searches on the same dataset (Nyquist frequency 50Hz) before and after applying bandpass filters. The first bandpass filter (bp: 1-20Hz) is selected as most seismic signals are under 20Hz; the second (bp: 3-20Hz) is selected after manually looking at sample spectrograms of the dataset and excluding noisy low frequencies. Figure 6.11 reports the similarity search runtime for fingerprints generated with different bandpass filters. Overall, similarity search suffers from additional matches generated from the noisy frequency bands outside the interests of seismology. For example, at station OXZ, removing the bandpass filter leads to a 16× slow down in runtime and a 209× increase in output size.

We compare detection recall on 8811 catalog earthquake events for different bandpass filters. The recall for the unfiltered data (0-50Hz), the 1-20Hz and 3-20Hz bandpass filters are 20.3%, 23.7%,

Table 6.2: The table shows that the percentage of fingerprints filtered (Filtered) and the false positive rate (FP) both increase as the occurrence filter becomes stronger (from filtering matches above 5.0% to above 0.1%). The runtime (in hours) measures similarity search time.

	LTZ (1548 events)			**MQZ** (1544 events)			**KHZ** (1542 events)		
Thresh	FP	Filtered	Time	FP	Filtered	Time	FP	Filtered	Time
>5.0%	0	0.09	149.3	0	0	2.8	0	0	2.2
>1.0%	0	30.1	31.0	0	0	2.7	0	0	2.3
>0.5%	0	31.2	32.1	0	0.09	2.8	0	0	2.4
>0.1%	0	32.1	28.6	0.07	0.3	2.7	0	0.03	2.4

	THZ (1352 events)			**OXZ** (1248 events)		
Thresh	FP	Filtered	Time	FP	Filtered	Time
>5.0%	0	0	2.4	0	0	2.6
>1.0%	0	0	2.3	0	0	2.6
>0.5%	0	0	2.4	0.08	0.08	2.7
>0.1%	0	0.02	2.3	0.08	0.17	2.6

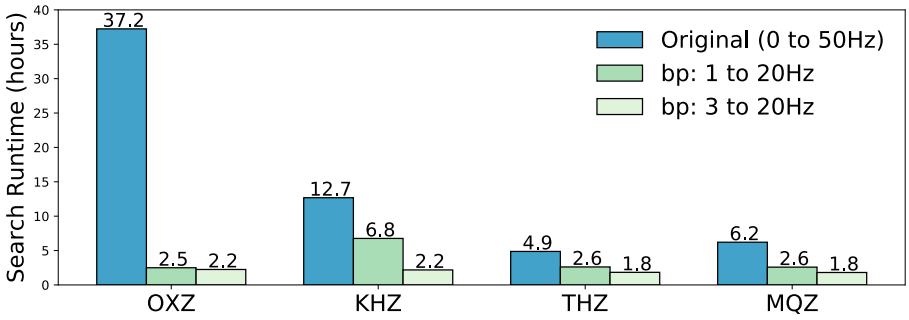


Figure 6.11: Effect of band pass filters on LSH runtime. Matches of noise in the non-seismic frequency bands can lead to significant increase in runtime for unfiltered time series.

45.2%, respectively. The overall low recall is expected, as we only used 4 (out of over 50) stations in the seismic network that contributes to the generation of catalog events. Empirically, a narrow, domain-informed bandpass filter focuses the comparison of fingerprint similarity only on frequencies that are characteristics of seismic events, leading to improved similarity between earthquake events and therefore increased recall. We provide guidelines for setting the bandpass filter in Appendix C of the extended report [270].

6.5.3 Effect of System Parameters

In this section, we evaluate the effect of core parameters on the system's performance and quality.

Sampling Rate	Accuracy (%)	Speedup
0.001	94.9	350×
0.01	98.7	99.8×
0.1	99.5	10.5×
0.5	99.7	2.2×
0.9	99.9	1.1×

Table 6.3: Speedup and quality of different MAD sampling rate compared to no sampling on 1.3M fingerprints. Sampling enables a 100x speed up in MAD calculation with 98.7% accuracy. Below 1%, runtime improvements suffer from a diminishing return, as the IO begins to dominate the MAD calculation in runtime.

MAD sampling rate. We evaluate the speed and quality trade-off for calculating the median and MAD of the wavelet coefficients for fingerprints via sampling. We measure the runtime and accuracy on the 1 month dataset in Section 6.5.1 (1.3M fingerprints) under varying sampling rates. Table 6.3 reports the relative speed up in MAD computation time and an accuracy metric that measures the average overlap between the binary fingerprints generated using the sampled MAD and the original MAD. Overall, runtime and accuracy both decrease with sampling rate, as expected. For example, a 10% and 1% sampling rate produce fingerprints with 99.7% and 98.7% accuracy, while enabling a near linear speedup of 10.5× and 99.8×, respectively. Below 1%, runtime improvements suffer from a diminishing return, as I/O begins to dominate MAD computation runtime.

LSH parameters. We report runtime of the similarity search under different LSH parameters in Figure 6.12. As indicated in Figure 6.6, the three sets of parameters that we evaluate yield near identical probability of detection given Jaccard similarity of two fingerprints. However, by increasing the number of hash functions and thereby increasing the selectivity of hash signatures, we decrease the average number of lookups per query by over 10x. This results in around 10x improvement in similarity search time.

Number of partitions. We report the runtime and memory usage of the similarity search with varying number of partitions in Figure 6.13. As the number of partitions increases, the runtime increases slightly due to the overhead of initialization and deletion of hash tables. In contrast, memory usage decreases as we only need to keep a subset of the hash signatures in the hash table at any time. Overall, by increasing the number of partitions from 1 to 8, we are able to decrease the memory usage by over 60% while incurring less than 20% runtime overhead. This allows us to run LSH on larger datasets with the same amount of memory.

Parallelism. Finally, to quantify the speedups from parallelism, we report the runtime of LSH hash signature generation and similarity search using a varying number of threads. For hash signature generation, we report time taken to generate hash mappings as well as the time taken to compute Min-Max hash for each fingerprint. For similarity search, we fix the input hash signatures and vary

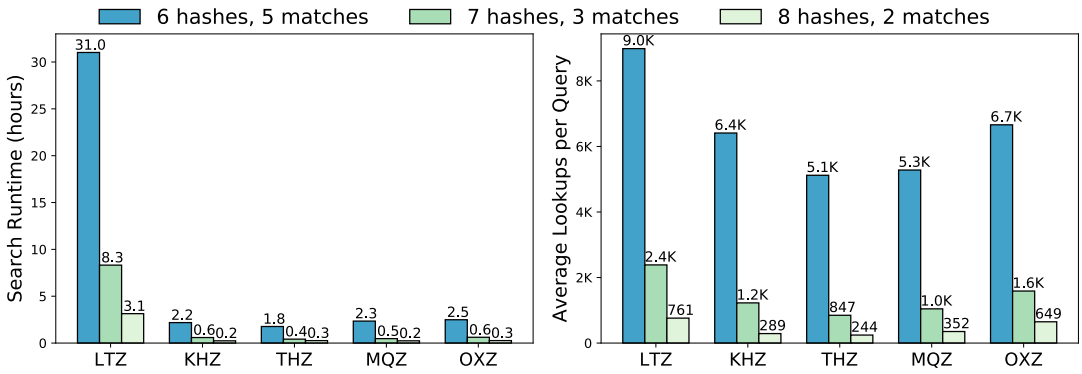


Figure 6.12: Effect of LSH parameters on similarity search runtime and average query lookups. Increasing the number of hash functions significantly decreases average number of lookups per query, which results in a 10× improvement in runtime.

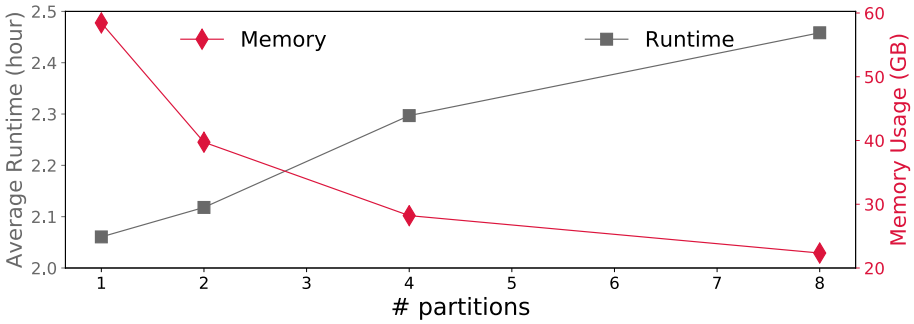


Figure 6.13: Runtime and memory usage for similarity search under a varying number of partitions. By increasing the number of search partitions, we are able to decrease the memory usage by over 60% while incurring less than 20% runtime overhead.

the number of threads assigned during the search. We show the runtime averaged from four seismic stations in Figure 6.14. Overall, hash signature generation scales almost perfectly (linearly) up to 32 threads, while similarity search scales slightly worse; both experience significant performance degradation running with all available threads.

6.5.4 Comparison with Alternative Similarity Search Algorithms

In this section, we compare the single-core query performance of our MinHash LSH to 1) an alternative open source LSH library FALCONN [112] and 2) state-of-the-art set similarity join algorithms: PPJoin [332], GroupJoin [49], AllPairs [36] and AdaptJoin [322]. We use 74,795 fingerprints with dimension 2048 and 10% non-zero entries, and a Jaccard similarity threshold of 0.5 for all libraries.

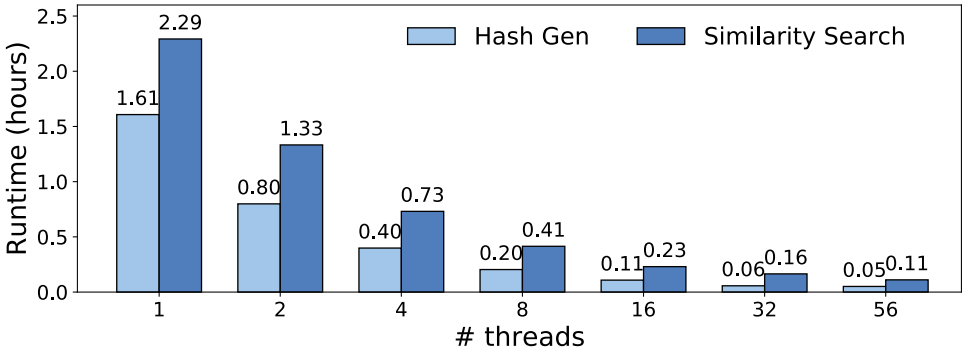


Figure 6.14: Hash generation scales near linearly up to 32 threads.

Table 6.4: Single core per-datapoint query time for LSH and set similarity joins. MinHash LSH incurs a 6.6% false negative rate while enabling up to 197× speedup.

Algorithm	Average Query time	Speedup
MinHash LSH	36 μs	–
FALCONN vanilla LSH [112]	.87ms	24×
FALCONN multi-probe LSH [112]	2.4ms	65×
AdaptJoin [322]	2.3ms	63×
AllPairs [36]	7.1ms	197×
GroupJoin [49]	5.7ms	159×
PPJoin [332]	5.5ms	151×

Exact Similarity Search Algorithms. We first investigate the performance and accuracy trade-off between using MinHash LSH and exact algorithms for similarity search. We focus the comparison on set similarity joins, a line of exact join algorithms that identifies all pairs of sets above a similarity threshold from two collections of sets [218]. State-of-the-art set similarity joins avoid exhaustively computing all pairs of set similarities via a filter-verification approach, such that only "promising" candidates that survive the filtering and verification are examined for the final join.

We report single-core query time of our MinHash LSH implementation and four state-of-the-art algorithms for set similarity joins: PPJoin [332], GroupJoin [49], AllPairs [36] and AdaptJoin [322]. For the set similarity joins, we use an open-source implementation (C++) from a recent benchmark paper, which is reported to be faster than the original implementations on almost all data points tested [218]. We transform each binary fingerprint into a set of integer tokens of the non-zero entries, with the tokens chosen such that larger integer tokens are more frequent than smaller ones.

We found that with a Jaccard similarity threshold of 0.5, the MinHash LSH incurs a 6.6% false negative rate while enabling 63× to 197× speedups compared to set similarity join algorithms (Table 6.4). Among the four tested algorithms, AdaptJoin achieves the best query performance as

False Negative (%)	Query time (ms)	# Hash Tables	# Probes
6.7	0.87	85	85
6.5	2.4	50	120
0.54	2.4	50	400
0.36	2.0	200	200

Table 6.5: Average query time and false negative rate under different FALCONN parameter settings.

a result of the small candidate set size enabled by its sophisticated filters. This is different from the benchmark paper's observation that expensive filters do not pay off and often lead to the slowest runtime [218]. One important difference in our experiment is that the input fingerprints have a fixed number of non-zero entries; as a result, the corresponding input sets have equal length. Therefore, filtering and pruning techniques based on set length do not apply to our dataset.

Alternative LSH Library. Next, we compare the query performance of our similarity search to an alternative and more advanced LSH library. We were unable to find an existing high-performance implementation of LSH for Jaccard similarity, so we instead compare it to FALCONN [112], a popular library based on recent theoretical advances in LSH family for cosine similarity [18].

We exclude hash table construction time, and compare single-core query time of FALCONN and our MinHash LSH. We use the cross-polytope LSH family and tune the FALCONN parameters such that the resulting false negative rate is similar to that of the MinHash LSH (6.6%). With "vanilla" LSH, FALCONN achieves an average query time of 0.87ms (85 hash tables); with multi-probe LSH, FALCONN achieves an average query time of 2.4ms (50 hash tables and 120 probes). In comparison, our implementation has an average query time of 36 μs (4 hash functions, 100 hash tables), which is 24× and 65× faster than FALCONN with vanilla and multi-probe LSH. We report the runtime and false negative rate under additional FALCONN parameter settings in Table 6.5. Notably, in multi-probe LSH, adding additional probes reduces the false negative rate with very little runtime overhead. We consider using multi-probe LSH to further reduce the memory usage as a valuable area of future work.

The performance difference reflects a mismatch between our sparse, binary input and FALCONN's target similarity metrics in cosine distance. Our results corroborate previous findings that MinHash outperforms SimHash on binary, sparse input data [289].

6.5.5 Comparison with Supervised Methods

In this section, we report results evaluating two supervised models on the Diablo Canyon dataset.

Models. We focus the evaluation on two supervised models: WEASEL [277] and ConvNetQuake [250]. The former is a time series classification model that leverages statistics tests to select discriminative bag-of-pattern features on Fourier transforms; it outperforms the state-of-the-art non-ensemble

classifiers in accuracy on the UCR time series benchmark. The latter is a convolutional neural network model with 8 strided convolution layers followed by a fully connected layer; it has successfully detected uncatalogued earthquakes in Central Oklahoma.

Data. Same as the qualitative study in Section 6.5.6, we focus on the area in the vicinity of the Diablo Canyon nuclear power plant in California. We use catalog earthquake events located in the region specified by Figure 6.17 as ground truth. We perform classification on the continuous ground motion data recorded at station PG.LMD, which has the largest number of high-quality recordings of catalog earthquake signals, and use additional data from station PG.DCD (station that remained active for the longest) for augmentation. Both stations record at 100Hz on 3 channels, capturing ground motion along three directions: EHZ channel for vertical, EHN channel for North-South and EHE channel for East-West motions. We use the vertical channel for WEASEL, and all three channels for ConvNetQuake.

Preprocessing and Augmentation. We extract 15-second long windows from the input data streams, which include windows containing earthquake events (positive examples) as well as windows containing only seismic noise (negative examples). This window length is consistent with that used for fingerprinting.

We adopt the recommended data preprocessing and augmentation procedures for the two models. For WEASEL, we z-normalize each 15-second window of time series by subtracting the mean and dividing by the standard deviation. For ConvNetQuake, we divide the input into monthly streams and preprocess each stream by subtracting the mean and dividing by the absolute peak amplitude; we generate additional earthquake training examples by perturbing existing ones with zero-mean Gaussian noise with a standard deviation of 1.2. For both models, we further augment the earthquake training set with examples of catalog events recorded at an additional station.

In order to prevent the models from overfitting to the location of the earthquake event in the time window (e.g. a spike in the center of the window indicates earthquakes), we generate 6 samples for each catalog earthquake event with the location of the earthquake event shifted across the window. Specifically, we divide the 15-second time window into five equal-length regions, and generate one training example from each catalog event with the event located at a random position within each region; we generate an additional example with an earthquake event located right in the center of the window. We report prediction accuracy averaged on samples located in each of the five regions for each event. We further analyze the impact of this augmentation in the results section below.

Train/Test Split. We create earthquake (positive) examples from the arrival times from the Northern California Seismic Network (NCSN) catalog [233]. Together, the catalog yields 3585 and 1388 catalog events for PG.LMD and PG.DCD, respectively, from 2007 to 2017. We select a random 10% of the catalog events from PG.LMD as the test set, which includes 306 events from 8 months. We create a second test set containing 449 new earthquake events detected by our system. Both test sets exhibit a similar magnitude distribution, with the majority of the events centered around magnitude

Table 6.6: Supervised methods trained on catalog events exhibit high false positive rate and a 20% accuracy gap between predictions on catalog and FASTer detected events.

	WEASEL [277]	**ConvNetQuake [250]**
Test Catalog Acc. (%)	90.8	90.6
Test FASTer Acc. (%)	68.0	70.5
True Negative Rate (%)	98.6	92.2
False Positive Rate (%)	90.0±5.88	90.0±5.88

1. The training set includes the remaining catalog events at PG.LMD, as well as additional catalog events at PG.DCD.

For negative examples, we randomly sample windows of seismic noise located between two catalog events at station PG.LMD. For training, we select 28,067 windows of noise for WEASEL, and 874,896 windows for ConvNetQuake; ConvNetQuake requires a much larger training set to prevent overfitting. For testing, we select 85,060 windows of noise from September, 2016 for both models.

Finally, we generate 15-second non-overlapping windows from one month of continuous data (December, 2011) in the test set. We then select 100 random windows that the model classifies as earthquakes for false positive evaluation.

Results. We report the two models' best classification accuracy on test noise events (true negative rate), catalog events and FASTer events in Table 6.6. The additional training data from PG.DCD boosts the classification accuracy for catalog and FASTer events by up to 4.3% and 3.2%. If the model is only trained on samples with the earthquake event in the center of the window, the accuracy further degrades to over 6% for WEASEL and over 20% for ConvNetQuake, indicating that the models are not robust to translation.

Overall, the 20% gap in prediction accuracy between catalog events and FASTer events suggests that models trained on the former do not generalize as well to the latter. Since the two test sets have similar magnitude distributions, the difference indicates that FASTer events might be sufficiently different from the existing catalog events in the training set that they are not detected effectively.

In addition, we report the false positive rate evaluated on a random sample of 100 windows predicted as earthquakes by each model. The ground truth is obtained via our domain collaborators' manual inspection. WEASEL and ConvNetQuake exhibit a false positive rate of 90% with a 95% confidence interval of 5.88%. In comparison, our end-to-end system has only 8% false positives. Figure 6.15 provides examples of the true positive and false positive predictions made by the ConvNetQuake model.

Discussion. The fact that unsupervised methods used in our system can find qualitatively different events than those in the existing catalog suggests that, for the earthquake detection problem, supervised and unsupervised methods are not mutually exclusive, but complementary to each other.

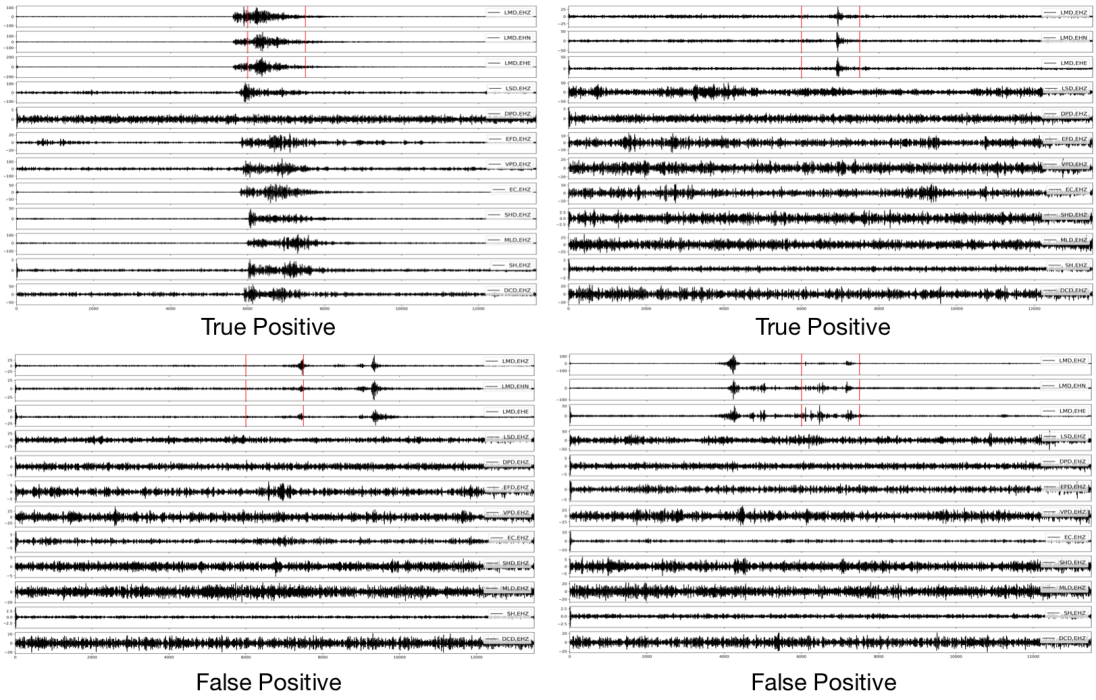


Figure 6.15: Example of true and false positive predictions made by the ConvNetQuake model.

In areas with rich historical data, supervised models showed promising potential for earthquake classification [250]. However, in cases where there are not enough events in the area of interest for training, we can still obtain meaningful detections via domain-informed unsupervised methods. In addition, unsupervised methods can serve as a means for label generation to improve the performance of supervised methods.

6.5.6 Qualitative Results

We first report our findings in running the system over a decade (06/2007 to 10/2017) of continuous seismic data from 11 seismic stations (27 total channels) near the Diablo Canyon nuclear power plant in central California. The chosen area is of special interest as there are many active faults near the power plant. Detecting additional small earthquakes in this region will allow seismologists to determine the size and shape of nearby fault structures, which can inform seismic hazard estimates.

We applied station-specific bandpass filters between 3 and 12 Hz to remove repeating background noise from the time series. In addition, we applied the occurrence filter on three out of the eleven seismic stations that experienced corrupted sensor measurements. The number of input binary fingerprints for each seismic channel ranges from 180 million to 337 million; the similarity search runtime ranges from 3 hours to 12 hours with 48 threads.

Among the 5048 detections above our detection threshold, 397 detections (about 8%) were false

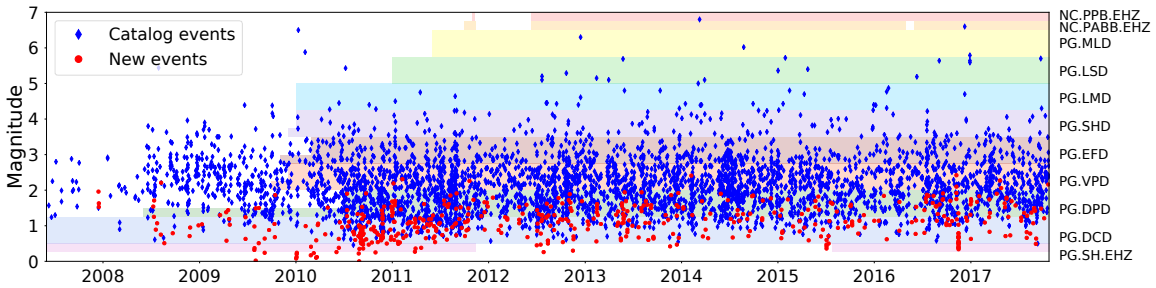


Figure 6.16: The left axis shows origin times and magnitude of detected earthquakes, with the catalog events marked in blue and new events marked in red. The colored bands in the right axis represent the duration of data used for detection collected from 11 seismic stations and 27 total channels. Overall, we detected 3957 catalog earthquakes (diamond) as well as 597 new local earthquakes (circle) from this dataset.

positives, confirmed via visual inspection: 30 were duplicate earthquakes with a lower similarity, 18 were catalog quarry blasts, 5 were deep teleseismic earthquakes (large earthquakes from >1000 km away). There were also 62 non-seismic signals detected across the seismic network; we suspect that some of these waveforms are sonic booms.

Overall, we were able to detect and locate 3957 catalog earthquakes, as well as 597 new local earthquakes. Figure 6.16 shows an overview of the origin time of detected earthquakes, which is spread over the entire ten-year span. The detected events include both low-magnitude events near the seismic stations, as well as larger events that are farther away. Figure 6.17 visualizes the locations of both catalog events and newly detected earthquakes, and Figure 6.18 zooms in on earthquakes in the vicinity of the power plant. Despite the low rate of local earthquake activity (535 total catalog events from 2007 to 2017 within the area shown in Figure 6.18), we were able to detect 355 new events that are between -0.2 and 2.4 in magnitude and located within the seismic network, where many active faults exist. We missed 261 catalog events, almost all of which originated from outside the network of our interest. Running the detection system at scale enables scientists to discover earthquakes from unknown sources. These new detected local events will be used to determine the details of active fault structures near the power plant.

We are also actively working with our domain collaborators on additional analysis of the New Zealand dataset. The system detected 11419 events, including 4916 catalog events, 355 teleseismic events, 6123 new local earthquakes and 25 false positives (noise waveforms) verified by the seismologists. We are preparing these results for publication in seismological venues, and expect to further improve the detection results by scaling up the analysis to more seismic stations over a longer duration of time.

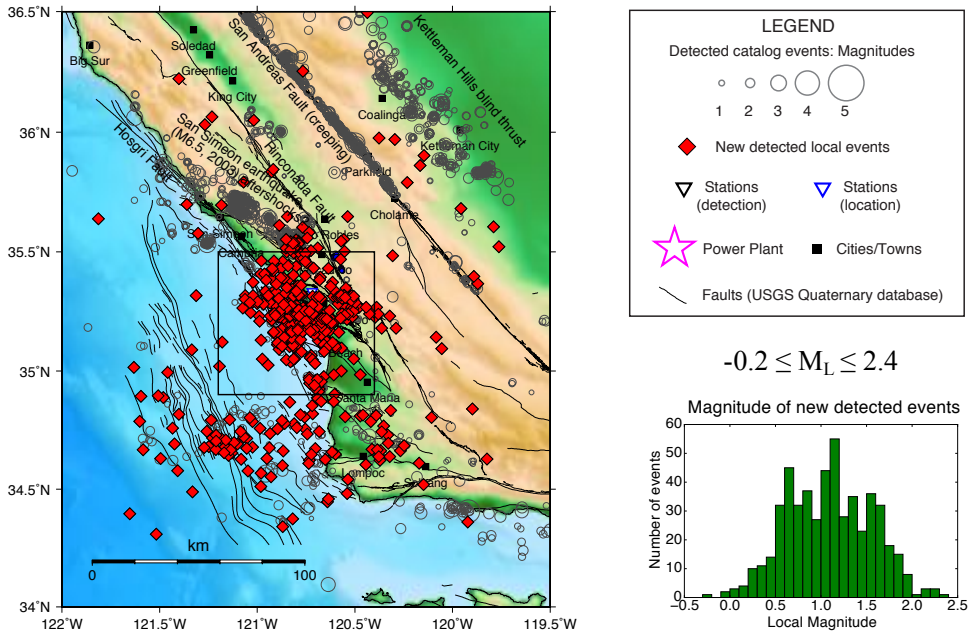


Figure 6.17: Overview of the location of detected catalog events (gray open circles) and new events (red diamonds). The system was able to detect earthquakes close to the seismic network (boxed) as well as all over California.

6.6 Discussion

In this section, we discuss related work and reflect on real-world usage of our system as well as lessons learnt from building it.

6.6.1 Related Work

First, we address related work in earthquake detection, LSH-based applications and time series similarity search.

Earthquake Detection. The original FAST work appeared in the seismology community, and has proven a useful tool in scientific discovery [107, 108]. In this paper, we present FAST to a database audience for the first time, and report on both the system composition and optimization from a computational perspective. The results presented in this paper are the result of over a year of collaboration between our database research group and the Stanford earthquake seismology research group. The optimizations we present in this paper and the resulting scalability results of the optimized system have not previously been published. We believe this represents a useful and innovative application of LSH to a real domain science tool that will be of interest to both the database community and researchers of LSH and time-series analytics.

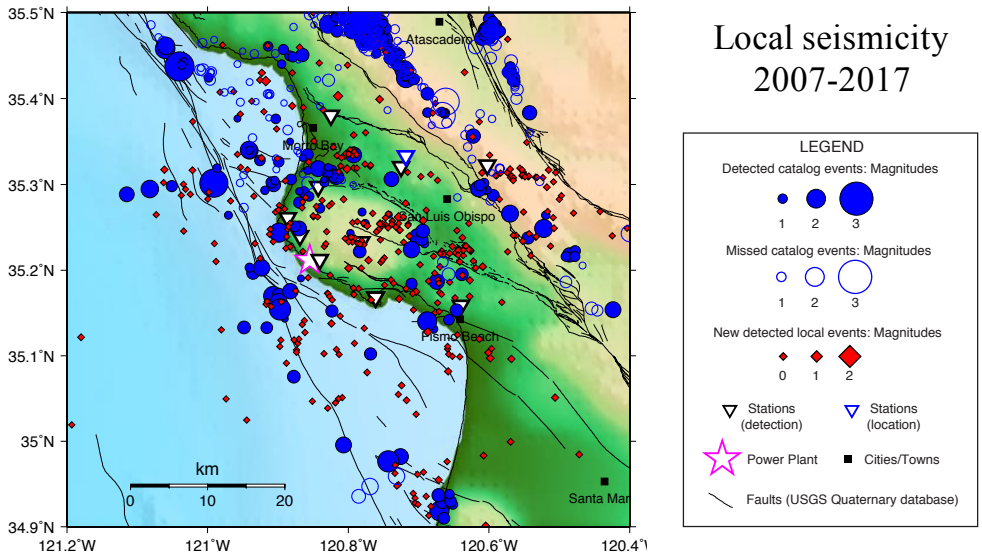


Figure 6.18: Zoom in view of locations of new detected earthquakes (red diamonds) and cataloged events (blue circles) near the seismic network (box in Figure 6.17). The new local earthquakes contribute detailed information about the structure of faults.

The problem of earthquake detection is decades old [16], and many classic techniques—many of which are in use today—were developed for an era in which humans manually inspected seismographs for readings [172, 328]. With the rise of machine learning and large-scale data analytics, there has been increasing interest in further automating these techniques. While FAST is optimized to find many small-scale earthquakes, alternative approaches in the seismology community utilize template matching [48, 279], social media [273], and machine learning techniques [24, 323]. Most recently, with sufficient training data, supervised approaches have shown promising results of being able to detect non-repeating earthquake events [250]. In contrast, our LSH-based detection method does not rely on labeled earthquake events and detects reoccurring earthquake events. In the evaluation, we compare against two supervised methods [250, 277] and show that our unsupervised method is able to detect qualitatively different events from the existing earthquake catalog.

Locality Sensitive Hashing. In this work, we perform a detailed case study of the practical challenges and the domain-specific solutions of applying LSH to the field of seismology. We do not contribute to the advance of the state-of-the-art LSH algorithms; instead, we show that classic LSH techniques, combined with domain-specific optimizations, can lead to scientific discoveries when applied at scale. Existing work shows that LSH performance is sensitive to key parameters such as the number of hash functions [105, 262]; we provide supporting evidence and analysis on the performance implication of LSH parameters in our application domain. In addition to the core LSH techniques, we also present nontrivial preprocessing and postprocessing steps that enable an end-to-end detection system, including spatiotemporal alignment of LSH matches.

Our work targets CPU workloads, complementing existing efforts that speed up similarity search on GPUs [171]. To preserve the integrity of the established science pipeline, we focus on optimizing the existing MinHash based LSH rather than replacing it with potentially more efficient LSH variants such as LSH forest [34] and multi-probe LSH [211]. While we share observations with prior work that parallelizes and distributes a different LSH family [308], we present the unique challenges and opportunities of optimizing MinHash LSH in our application domain. We provide performance benchmarks against alternative similarity search algorithms in the evaluation, such as set similarity joins [218] and an alternative LSH library based on recent theoretical advances in LSH for cosine similarity [18]. We believe the resulting experience report, as well as our open source implementation, will be valuable to researchers developing LSH techniques in the future.

Time Series Analytics. Time series analytics is a core topic in large-scale data analytics and data mining [186, 204, 333]. In our application, we utilize time series similarity search as a core workhorse for earthquake detection. There are a number of distance metrics for time series [102], including Euclidean distance and its variants [337], Dynamic Time Warping [260], and edit distance [319]. However, our input time series from seismic sensors is high frequency (e.g. 100Hz) and often noisy. Therefore, small time-shifts, outliers and scaling can result in large changes in time-domain metrics [59]. Instead, we encode time-frequency features of the input time series into binary vectors and focus on the Jaccard similarity between the binary feature vectors. This feature extraction procedure is an adaptation of the Waveprint algorithm [31] initially designed for audio data; the key modification made for seismic data was to focus on frequency features that are the most discriminative from background noise, such that the average similarity between non-seismic signals is reduced [41]. An alternative binary representation models time series as points on a grid, and uses the non-empty grid cells as a set representation of the time series [245]. However, this representation does not take advantage of the physical properties distinguishing background from seismic signals.

6.6.2 Usage and Reflection

In this work, we reported on a novel application of LSH to large-scale seismological data, as well as the challenges and optimizations required to scale the system to over a decade of continuous sensor data. This experience in scaling LSH for large-scale earthquake detection illustrates both the potential and the challenge of applying core data analytics primitives to data-driven domain science on large datasets. On the one hand, LSH and, more generally, time series similarity search, is well-studied, with scores of algorithms for efficient implementation: by applying canonical MinHash-based LSH, our seismologist collaborators were able to meaningfully analyze more data than would have been feasible via manual inspection. On the other hand, the straightforward implementation of LSH in the original FAST algorithm failed to scale beyond a few months of data. The particulars of seismological data—such as frequency imbalance in the time series and repeated background

noise—placed severe strain on an unmodified LSH implementation and on researchers attempting to understand the output.

As a result, the seismological discoveries we have described in this paper would not have been possible without domain-specific optimizations to the detection system. Our domain scientist collaborators at Stanford were also able to publish a detailed analysis of these new findings in the seismology journal Bulletin of the Seismological Society of America [338]. In addition, our open-source system [114] has already received interest from researchers worldwide, including McGill University, University College London, Dublin Institute for Advanced Studies, Ludwig Maximilian University of Munich, Austral University of Chile, and National Geophysical Research Institute in India.

We believe that these results have important implications for researchers studying LSH (e.g., regarding the importance of skew resistance) and will continue to bear fruit as we scale the system to even more data and larger networks.

Chapter 7

Discussion and Future Directions

We observed two bottlenecks that prevent data analytics systems from managing exponentially increasing data volumes. First, analytics systems can not afford to compute all ingested data, especially for computationally expensive tasks and applications requiring interactivity. Second, analysts cannot afford to manually inspect all the results generated by analytics systems to determine which merit further investigation.

In this dissertation, we presented systems and algorithms to improve computational and human efficiency in data analytics by focusing the limited computational resources and analysts' attention on a subset of relevant data. To improve computational efficiency, we demonstrated that novel combinations of precomputation and query-time sampling significantly improve query performance with small precomputation overheads in AQP systems (Chapters 3 and 4). To improve human efficiency, we proved that the automatic highlighting and summarization of important behaviors in data could save end-users' time in manual inspections and improve the usability of monitoring applications (Chapters 5 and 6). In this chapter, we discuss the limitations of our work, along with exciting directions for future work.

7.1 Limitations

Storage Considerations. Thus far, this dissertation has assumed that ingestion and storage are solved problems and instead focused on efficiently utilizing ingested data. Specifically, the assumptions were 1) that users can collect and store all needed data and 2) that network traversal times are not a bottleneck. While these assumptions held true for the specific tasks we targeted, they should be revisited in the broader context.

Users today can collect and store more data than ever due to improved storage technology and the availability of cloud storage services. For example, hard drive prices have dropped from over

$1,000,000 per gigabyte in 1981 to less than $0.02 per gigabyte in 2021 [222]. Similarly, since Amazon's initial release of S3 in 2006 at a monthly cost of $0.15 per gigabyte, major vendors such as Google, Microsoft, Oracle, and IBM have all started offering cloud storage services, driving the costs down to as low as $0.001 per gigabyte per month [15]. While the storage prices continue to decrease, the rate of change has slowed down. Since data volumes continue to grow exponentially, future data analytics systems need to put more thought into deciding which data to store and how to physically organize the data. For example, sensors at the Large Hadron Collider are currently generating so much data that scientists developed intelligent filters to select a small fraction of potentially pertinent events to retain.

Furthermore, while computing and storage have traditionally been collocated, there has been increasing interest in architectures that advocate the decoupling and disaggregation of storage and computing resources with improvements in network technology [148, 302, 320]. In these architectures, data can be located in remote storage, such as Amazon S3 or Azure Blob Storage, while all communications occur over the network. This setup is supported in PS^3, and even when data is stored remotely, network traversal times do not produce a bottleneck because our data accesses have no sequential dependencies. Namely, given a query, PS^3 knows all the partitions to request upfront. In contrast, if we were to implement a pointer-chase traversal algorithm in which each request depends on information gained from the previous one, the network time could begin to dominate. For desegregated architectures, our approach of leveraging lightweight precomputation to reduce the amount of data shipped over the network can benefit compute-intensive applications. However, for other applications where network time is a concern, it could be more beneficial to offload some computation to storage, such as via stored procedures and user-defined functions.

Domain-Specific Optimizations. This dissertation has repeatedly illustrated the importance of domain knowledge and domain-specific optimizations. In ASAP, we leveraged the fact that the granularity of the visualization is limited by the resolution supported by the display device to significantly reduce the computational complexity of the search for smoothing parameters. In FASTer, we leveraged knowledge of earthquake events both to reduce repeating background noise patterns, thus improving the efficiency of the similarity search procedure, and to aggregate similar search results across multiple sensors to improve the confidence of the results presented to users. We have incorporated domain knowledge by directly working with domain experts and hand engineering specific optimizations into the system. Although these optimizations achieved excellent results in individual systems, manually developing optimizations is not scalable, as each domain and application can require completely different solutions.

At least two missing components prevent the analytics systems from automatically encoding users' domain expertise into the analysis pipeline. The first is the lack of an intuitive interface or language that defines the search space of the optimization. Query languages played a crucial role in the development of relational databases. Each query defines the list of possible actions, such

as accessing data in different orders or using different implementations of an operator. Similarly, modern analytics workloads contain many operator and parameter choices for feature transformation, data modeling, and statistical analyses. Defining a scope that is simultaneously broad enough to support multiple applications and narrow enough to share similar query and data characteristics for optimizations is the first step towards automation. The second component is the lack of mechanisms to solicit user feedback. Unlike in traditional query optimization, it is essential to leverage domain knowledge to improve both the query performance and the result quality of the analysis. While it is relatively easy to collect statistics on the former, the latter requires explicit user feedback. Developing strategies to solicit the most information from users with the least effort is essential in this iterative development process.

7.2 Future Directions

7.2.1 Improving Efficiency

In this dissertation, we improved query processing efficiency by leveraging knowledge of the workload and application to select a subset of stored data to process intelligently. As data volumes continue to increase, it is essential to consider how to push this application semantics further down the data analytics pipeline, such as during the ingestion and storage phases. This section describes two ideas for future research.

Self-organizing Data Layout. Partition pruning based on summary statistics (e.g., the range of values in each column) is widely used in current analytics systems. Using this approach, queries can skip reading data partitions if the file metadata indicates that the file contains no matching values. Therefore, data layouts, or how individual data points are assigned to data blocks, have a major impact on query performance. Existing work on data layout optimization has focused on the offline setup, which assumes knowledge of the query workload and aims to design a partitioning scheme for the dataset that maximizes the amount of data skipped under this workload [306, 334]. However, in practice, users might not have prior knowledge of the workload, and repartitioning the entire dataset each time new queries arrive is expensive. Many opportunities lie in designing storage systems that can incrementally optimize their layouts according to changing data and query distributions in an online fashion. For example, Snowflake has already implemented heuristics-based policies for incremental partition maintenance.

The idea of self-organizing storage has been explored in database cracking, which physically reorganizes a column incrementally as each query arrives [336]. The cost of cracking decreases as more queries arrive, since the strategy enforces a sort order on the column in the limit. However, the reorganization cost does not decrease over time in our case; shifts in query workloads can lead to entirely different partition designs. Therefore, the online layout optimization problem requires

new strategies that can dynamically trade off the costs of repartitioning with the gains in query performance to determine when and which part of the data to reorganize. It remains to be seen how well such an adaptive strategy would perform compared to a static layout with access to the entire query workload.

Specialized Compression for Time-Series Data. Compression not only reduces storage costs but also increases query throughput by allowing more data to fit into the memory. Time-series data can benefit significantly from compression techniques due to its high volumes (e.g., 2.5 terabytes per second at Google [6]). However, generic compression formats, such as gzip, fail to leverage the unique properties of time series data, such as temporal correlations between consecutive samples in a measurement and between different measurements of the same entity. For example, if the values do not change significantly between consecutive measurements, it is possible to store the XOR of the differences between the two values instead [244]. In addition, common analytics operations on time series such as aggregations are agnostic to the ordering of data points.

Some ideas for incorporating these unique properties of time series data and its workloads into the storage and organization of analytics systems include storing correlated time series together, which can lead to better compression ratios but requires decompressing additional data during query time. Another approach could be to leverage query patterns and the natural hierarchies in the data specified by associated categorical attributes (e.g., region→data center→rack) to group time series and achieve a balance between the compression ratio and query performance. Recent time-series data is queried more often and at a higher granularity compared to historical data. Accordingly, another solution would be to design systems that dynamically adjust the granularity of data stored, provided it can be done cheaply and without undergoing the entire process of decompression, aggregation, and compression. The Burrows–Wheeler transform rearranges a string into runs of repeated characters to achieve a better compression ratio using techniques such as run-length encoding. Since floating-point numbers that are close in value can be better compressed using the XOR technique and since common aggregation operations are agnostic to the ordering of data points, the time series could be reordered to achieve a better compression ratio analogous to the Burrows-Wheeler transform.

7.2.2 Improving Usability

Through this dissertation, we have improved the usability of monitoring applications by automatically identifying, highlighting, and summarizing important behaviors in large data volumes. This improved usability is reflected in the real-world usages and deployments of ASAP and FASTer. However, the progress largely resulted from hand-engineered, domain-specific adaptations and optimizations. In this section, we provide some ideas for generalizing lessons learned and applying them to a broader set of use cases and users.

Visualization and Data Exploration. ASAP demonstrates the power of summary visualizations: they hide distracting details in complex datasets and help focus users' attention on the high-level properties of the datasets. However, ASAP's scope is limited: we specifically picked moving average as the smoothing function and chose to preserve outlyingness in the problem formulation to support the monitoring use case. Many summary techniques for visualization exist in the broader design space, including sampling, filtering, clustering, and aggregation. There are also additional properties in the dataset that users might value outside of anomalies, such as trend, cluster, frequency, correlation, and structure.

Many questions remain related to building interactive visual analytics systems to help users explore and debug large datasets with summary visualizations. In terms of usability, it is unclear which visualization technique is best suited for a given use case and dataset. In addition, summary visualizations could introduce biases by hiding details and only presenting certain aspects of the datasets. It is important to identify these potential risks and design corresponding mechanisms to correct them. In terms of performance, achieving interactivity on large datasets is non-trivial; prior work has demonstrated that neither fast query engines nor traditional AQP techniques are sufficient [56,191]. Promising results that leverage cached samples, specialized visualization indexes and sketches, and incremental computation in interactive data analyses are already evident, but more questions remain than answers.

7.3 Closing Thoughts

We are in an exciting era for data. Sensors and devices have generated unprecedented volumes of data, while cloud solutions have significantly reduced storage costs. However, data is meaningless until it gets processed and interpreted. From ingestion to computation to interpretation, each step in the data analytics pipeline becomes more expensive and can handle much smaller data volumes. To enable the computational resources and analysts' attention to keep up with the fast increasing data volumes, this dissertation introduced systems and algorithms that focus the limited resources on a small subset of relevant data and behaviors.

Although this dissertation presented computation and analysts' attention as two separate bottlenecks, humans and machines go hand-in-hand in analytics. Efficient computation primitives are key to reducing delays and improving user productivity in exploratory analyses. Users' preferences and knowledge, in turn, introduce unique optimization opportunities for the computation. As our community moves forward to support increasingly complex analytics workflows that combine data integration and wrangling, statistical analysis, machine learning, and visualization, it is important to not only focus on challenges in individual stages of the workflow, but also develop end-to-end systems and tools that help real users and use cases.

Appendix A

Supplementary material for PS^3

A.1 Implementation Details

In this section, we provide additional implementation details for the partition picker.

A.1.1 Clustering

Normalization. Prior to clustering, we normalize the summary statistics to make sure that the euclidean distance is not dominated by any single statistic. We first apply a log transformation to reduce the overall skewness to all summary statistics except for selectivity estimates; for the selectivity estimates which are between 0 and 1, we use the cube root transformation instead. We then normalize each summary statistics by its average value in the training dataset. We choose the average instead of the max as the normalization factor since it is more robust to outliers. During test time, the statistics are normalized by their corresponding average values in the training dataset.

Feature Selection. We provide pseudo code for the feature selection procedure in Algorithm 13.

We report the features selected by the procedure on the four real-world datasets for experiments reported in § 3.4.2:

- `TPC-H*`: selectivity_upper, selectivity_lower, min(x), max hh, max dv, hh_bitmap
- `TPC-DS*`: $\log^2(x)$, $\overline{x}$, sum dv, hh_bitmap
- `Aria`: selectivity_indep, selectivity_max, min($\log(x)$), $\overline{x}$, max(x), avg hh, # dv
- `KDD`: selectivity_indep, $\overline{x^2}$, max dv

Only a small number of features are used in each dataset, but across datasets, all four types of features are represented. This again illustrates the need for all four sketches.

Finally, we measure the quantitative impact of the feature selection procedure on clustering performance in Table A.1. Similar to the experiment in §3.4.5, we evaluate the *average relative*

Algorithm 13 Feature Selection for Clustering

```
feats ← (selectivity, occurrence_bitmap,
         log(x), log^2(x), min(log(x)), max(log(x)),
         x̄, x̄^2, std, min(x), max(x),
         # hh, max hh, avg hh,
         # dv,avg dv, max dv, min dv, sum dv)
best ← []                                   ▷ Features excluded from clustering
for i ← 1 → 10 do
    feats.shuffle()                         ▷ Explore features in random order
    to_exclude ← []
    for f ∈ feats do
        new ← [to_exclude]+[f]
        if ImproveCluster(to_exclude, new) then
            to_exclude ← new
        end if
    end for
    if ImproveCluster(best, to_exclude) then
        best ← to_exclude
    end if
end for
return best
```

Table A.1: Area under the curve for the average relative error of clustering under different sampling budgets for Hierarchical Agglomerative Clustering (HAC) and KMeans clustering; smaller is better.

	HAC (ward)	+feat sel	KMeans	+feat sel
TPCDS	4.2	3.8 (-9%)	4.2	3.8 (-8%)
Aria	2.6	2.3 (-14%)	2.7	2.3 (-15%)
KDD	.58	.55 (-5%)	.55	.54 (- .5%)

error for estimating the query answer using different clustering procedures, and compare the total area under the error curve for different sampling budgets. Overall, feature selection consistently improves clustering performance for both clustering methods, reducing the area from 0.5% to 15% across datasets.

A.1.2 Training

We use the `XGBoost` regressor as our base model and use the squared error as the loss function. Although our models are only used for binary classification, we train them as regressors instead of classifiers. This is to address the problem that the ratio of positive to negative examples are different for different queries. Consider a query which has one partition with rows that satisfy the predicate versus a query with 100 such partitions. Missing one positive example would have a much larger impact on the final accuracy for the first query compared to the second. While a classifier can only handle class imbalance globally, with a regressor, we can scale labels differently such that

Algorithm 14 Training Label Generation

Input: threshold $t \in [0, 1]$, partition count n, feature dimension m, query answer dimension d; for each input query i, partition features $F_i \in \mathbb{R}^{n \times m}$ and normalized query answers on each partition $A_i \in [0, 1]^{n \times d}$

Output: X, Y

$X \leftarrow [], Y \leftarrow []$
for each $(F_i, A_i) \in$ training **do** ▷ For each query
  ans $\leftarrow \sum(A_i)$ ▷ Ground truth query answer
  for $j \leftarrow 1 \rightarrow n$ **do**
    $y[j] \leftarrow \max(A_i[j]) > t$ ▷ Partition contribution
  end for
  positive $\leftarrow \sum y$
  for $j \leftarrow 1 \rightarrow n$ **do**
    if $y[j] == 1$ **then**
      $y[j] \leftarrow \sqrt{\frac{c}{positive}}$
    else
      $y[j] \leftarrow -\sqrt{\frac{c}{n-positive}}$
    end if
  end for
  $X.append(F_i)$
  $Y.append(y)$
end for

the positive examples weigh more in the first query. We provide pseudo code for the training set up in Algorithm 14.

A.2 Variance Analysis

A.2.1 Unbiased picker

Unbiased picker. We introduce an unbiased version of our proposed estimator that lends well to analysis. As described in § 3.3.2, the biased estimator picks an exemplar partition deterministically from a cluster given the median feature vector of the cluster, whereas the unbiased estimator picks a cluster exemplar partition at random. We empirically compare the performances of the two estimators on four real-world datasets in Figure A.1. For each test query, we run the unbiased estimator 10 times and compute the average error achieved to compare against the error achieved by the biased estimator.

Overall, Figure A.1 shows that the biased estimator achieves smaller error compared to the unbiased version when the sampling fraction is small, and that there are no significant differences in accuracy between the two estimators otherwise. In addition, for a given query, the biased version of the estimator has *no variance*. Therefore, in use cases when the sampling budget is limited or when

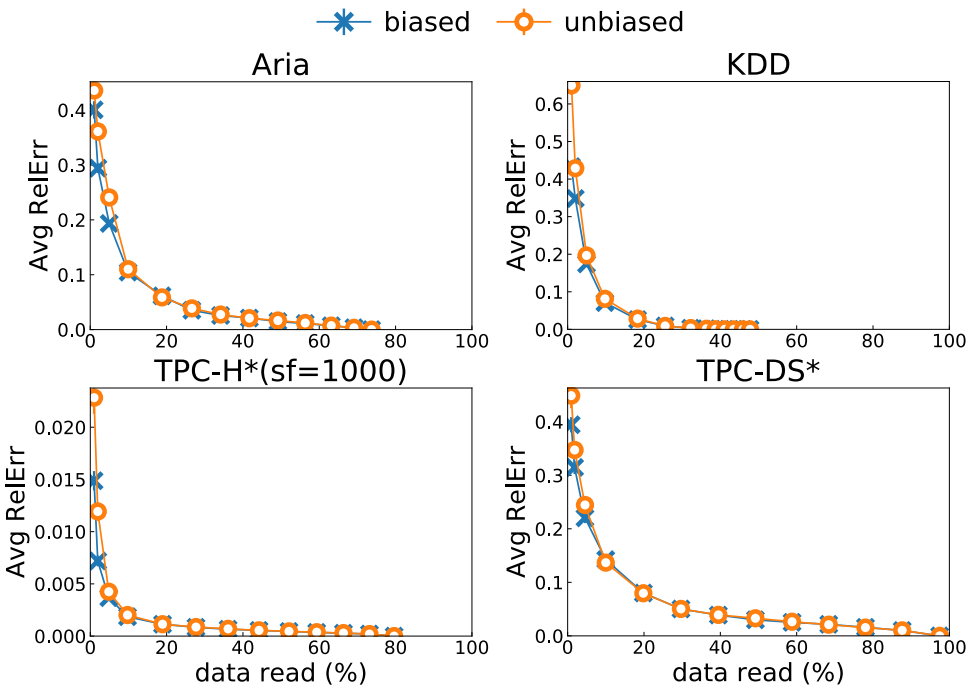


Figure A.1: Empirical comparison of the bias and unbiased version of the estimator. The biased estimator tends to outperform the unbiased when the sampling fraction is small.

users prefer getting a deterministic answer for a given query, the biased version of the estimator might be preferred.

Analysis. Next, we analyze the unbiased version of the estimator using the framework of stratified sampling. Compared to a simple random sample of the same size, stratified sampling can produce an estimator with smaller variance if the elements within strata are homogeneous. In our case, each cluster is essentially a stratum; if clustering is effective, the partitions in a cluster are similar to each other, leading to a variance reduction.

Within each stratum, we perform simple random sampling without replacement (SRSWoR) to draw a sample of size 1; the variance formula for SRSWoR can be found in Chapter 2.5.2 of [83]. Note that since we only draw one sample from each cluster/stratum, in order to estimate variance of the stratum, we would need to evaluate *additional* partitions per stratum. Finally, the total variance of the unbiased estimator is the sum of the variances from each stratum.

When central limit theorem holds, the 95% confidence interval of an estimator Y is given by $\pm 1.96\sqrt{\sigma^2(Y)}$ [83], where $\sigma^2(Y)$ is the variance of the estimator described above.

A.2.2 Partition-level v.s. Row-level Sampling

In this subsection, we compare random partition level sampling to random row level sampling. We show that under the same sampling fraction, random partition level sampling has much larger variance than random row level sampling.

Set up. We start with a description of the setup. For a group G in the query, let y_i be the value of the aggregate function on partition i. Let π_i be the probability that partition i is included in the sample, π_{ij} be the probability that both partition i and j are in the sample, N be the total number of partitions and S be the set of sampled partitions.

We wish to estimate the total value of the aggregate function for group G on all partitions. For `SUM` and `COUNT` queries, the total value is $Y = \sum_{i=1}^{N} y_i$. If all partitions have positive sampling probability ($\pi_i > 0, \forall i$), an unbiased *Horvitz-Thompson* estimator for Y under Poisson sampling is:

$$\hat{Y} = \sum_{i \in S} \frac{y_i}{\pi_i}$$

The true variance of the estimator $\hat{Y}$ is:

$$\sigma^2(\hat{Y}) = \sum_{i,j=1}^{N} (\frac{\pi_{ij}}{\pi_i \pi_j} - 1) y_i y_j \tag{A.1}$$

However, since y_i is only available for partitions that are included in the sample, we can not evaluate the true variance using Eq A.1 directly. Instead, we estimate the true variance using the sampled set of partitions S [83]:

$$\hat{\sigma}^2(\hat{Y}) = \sum_{i,j=1}^{N} (\frac{1}{\pi_i \pi_j} - \frac{1}{\pi_{ij}}) y_i y_j \tag{A.2}$$

If the second-order inclusion probability $\pi_{ij} > 0$ for all pairs of partitions i, j, Eq A.2 is an unbiased estimator for Eq A.1, the true variance of $\hat{Y}$ [121].

Analysis. For random partition level sampling, assume that each partition is selected in the sample with probability p. The expected size of S is Np. Since the partitions are sampled independently, $\pi_{ij} = \pi_i \pi_j$. Plug the inclusion probabilities in Eq A.2, the estimator of the true variance is:

$$\hat{\sigma}^2(Y_{blk}) = \sum_{i \in S} (\frac{1}{p^2} - \frac{1}{p}) y_i^2 \tag{A.3}$$

Similarly, assume that each tuple is sampled with probability p. Let t_x be the total value that a tuple x contributes towards the aggregate for group G, and S_t be the set of sampled tuples. Following

similar derivation as Eq A.3, the estimator of the variance for random row level sampling is

$$\hat{\sigma}^2(T_{row}) = \sum_{x \in S_t} (\frac{1}{p^2} - \frac{1}{p}) t_x^2 \tag{A.4}$$

Note that y_i in Eq A.3 is simply the sum of tuples in partition i. Let b_x be the partition that contains tuple x, then $y_i = \sum_{b_x=i} t_x$. Therefore,

$$y_i^2 = \sum_{b_x=i} t_x^2 + 2 \sum_{\substack{x<y, \\ b_x=b_y=i}} t_x t_y$$

Eq A.3 can be rewritten as

$$\hat{\sigma}^2(T_{blk}) = \sum_{i \in S_t} (\frac{1}{p^2} - \frac{1}{p}) t_i^2 + 2 \sum_{\substack{i,j \in S_t, \\ i<j, b_i=b_j}} (\frac{1}{p^2} - \frac{1}{p}) t_i t_j \tag{A.5}$$

Comparing to random row-level sampling with the same sampling fraction p (Eq A.4), random partition-level sampling has larger variance: Eq A.5 includes an additional term that accounts for the variance contributed by tuples belonging to the same partition.

Appendix B

Supplementary material for HBE

B.1 Proofs

B.1.1 Preliminaries

Basic inequalities. We first state without proof some well known inequalities that we will use in the proofs.

Lemma 6 (Chebyshev's and Paley-Zygmund inequalities)**.** *For a non-negative random variable Z and parameters $t > 0$, $\theta \in [0, 1]$, we have*

$$\mathbb{P}[Z \geq (t+1) \cdot \mathbb{E}[Z]] \leq \frac{1}{t^2} \cdot \mathrm{RelVar}[Z], \tag{B.1}$$

$$\mathbb{P}[Z > (1-\theta)\mathbb{E}[Z]] \geq \frac{1}{1 + \frac{1}{\theta^2} \cdot \mathrm{RelVar}[Z]}. \tag{B.2}$$

Theorem 7 (Chernoff bounds)**.** *Let $X = \sum_{i=1}^{n} X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i are independent. Let $v = \mathbb{E}[X] = \sum_{i=1}^{n} p_i$. Then for $\delta > 0$*

$$\mathbb{P}[X \geq (1+\delta)v] \leq e^{-\frac{\delta^2}{2+\delta}v}, \tag{B.3}$$

$$\mathbb{P}[X \leq (1-\delta)v] \leq e^{-\frac{1}{2}\delta^2 v}. \tag{B.4}$$

Median-trick to boost success probability. The median-trick is based on concentration of sums of independent binary random variables. If we define binary random variables appropriately we can obtain bounds for the concentration of the median of i.i.d. random variables around their expectation.

Lemma 7. *Let $Z_1, \ldots, Z_L$ be $L \geq 1$ i.i.d. copies of a non-negative random variable with* $\mathrm{RelVar}[Z] \leq$

$\frac{\epsilon^2}{6}$ *then:*

$$\mathbb{P}\left[\text{median}\{Z_1,\ldots,Z_L\} \geq (1+\epsilon)\mathbb{E}[Z]\right] \leq e^{-\frac{L}{6}},$$
$$\mathbb{P}\left[\text{median}\{Z_1,\ldots,Z_L\} \leq (1-\epsilon)\mathbb{E}[Z]\right] \leq e^{-\frac{L}{4}}.$$

Proof of Lemma 7. Let

$$X_i = \mathbb{I}[Z_i \geq (1+\epsilon)\mathbb{E}[Z]],$$
$$Y_i = \mathbb{I}[Z_i \leq (1-\epsilon)\mathbb{E}[Z]].$$

By Lemma 6, we have that

$$a_i = \mathbb{E}[X_i] \leq \frac{1}{\epsilon^2}\frac{\epsilon^2}{6} \leq \frac{1}{6}, \quad b_i = \mathbb{E}[Y_i] \leq \frac{1}{7}.$$

We get the following upper bounds

$$\mathbb{P}[\text{median}\{Z_1,\ldots,Z_L\} \geq (1+\epsilon)\mathbb{E}[Z]] \leq \mathbb{P}[\sum_{i=1}^{L} X_i \geq \frac{L}{2}].$$
$$\mathbb{P}[\text{median}\{Z_1,\ldots,Z_L\} \geq (1+\epsilon)\mathbb{E}[Z]] \leq \mathbb{P}[\sum_{i=1}^{L} Y_i \geq \frac{L}{2}],$$

that along with Chernoff bounds will give us our result. We only show the first inequality as the second one follows similarly. Let $A = \sum_{i=1}^{L} a_i \leq L/6$, the first event is bounded by $\exp(-\frac{(\frac{L/2}{A}-1)^2}{2+(\frac{L/2}{A}-1)}A) \leq \exp(-L/6)$. □

Moments of Hashing-Based-Estimators.

Lemma 8. *Assuming that* $\forall i \in [n], p(x_i, q) > 0$ *then*

$$\mathbb{E}[Z_h] = \sum_{i=1}^{n} u_i k(x, x_i), \tag{B.5}$$
$$\mathbb{E}[Z_h^2] = \sum_{i,j=1}^{n} k^2(q, x_i)\frac{u_i \mathbb{P}[i,j \in H(q)]u_j}{p^2(q, x_i)}. \tag{B.6}$$

Proof of Lemma 8. We start with the expectation:

$$\begin{aligned}
\mathbb{E}_{h,X}[\frac{k(q,X)}{p(q,X)}u_{H(q)}] &= \mathbb{E}_h[\mathbb{E}_X[\frac{k(q,X)}{p(q,X)}]u_{H(q)}] \\
&= \mathbb{E}_h[\sum_{i\in H(q)}\frac{u_i}{u_{H(q)}}\frac{k(q,x_i)}{p(q,x_i)}u_{H(q)}] \\
&= \sum_{i=1}^{n} u_i \mathbb{E}[\mathbb{I}[h(x_i)=h(q)]]\frac{k(x_i,q)}{p(x_i,q)} \\
&= \sum_{i=1}^{n} u_i k(x_i,q)
\end{aligned}$$

We proceed with the second moment:

$$\begin{aligned}
\mathbb{E}_{h,X}[\frac{k^2(q,X)}{p^2(q,X)}u^2_{H(q)}] &= \mathbb{E}_h[\mathbb{E}_X[\frac{k^2(q,X)}{p^2(q,X)}]u^2_{H(q)}] \\
&= \mathbb{E}_h[\sum_{i\in H(q)}\frac{u_i}{u_{H(q)}}\frac{k^2(q,x_i)}{p^2(q,x_i)}u^2_{H(q)}] \\
&= \mathbb{E}_h[\sum_{i\in H(q)} u_i\frac{k^2(q,x_i)}{p^2(q,x_i)}u_{H(q)}] \\
&= \mathbb{E}_h[\sum_{i,j\in H(q)} u_i u_j\frac{k^2(q,x_i)}{p^2(q,x_i)}] \\
&= \sum_{i,j=1}^{n} k^2(x_i,q)\frac{u_i\mathbb{P}[i,j\in H(q)]u_j}{p^2(x_i,q)}
\end{aligned}$$

□

B.1.2 Refined Variance bound

Here, we derive our new inequality bounding the variance of HBE and RS. Let $\mu \le \lambda \le L \le 1$ and define:

$$S_1 = \{i \in [n] : L \le w_i \le 1\} \tag{B.7}$$

$$S_2 = \{i \in [n] \setminus S_1 : \lambda \le w_i \le L\} \tag{B.8}$$

$$S_3 = \{i \in [n] \setminus (S_2 \cup S_1) : \mu \le w_i \le \lambda\} \tag{B.9}$$

$$S_4 = \{i \in [n] : w_i < \mu\} \tag{B.10}$$

as well as $\mu_\ell = \sum_{i\in S_\ell} u_i w_i \le \mu$. The intuition behind the definition of the sets is that for radial decreasing kernels they correspond to spherical annuli around the query (Figure B.1).

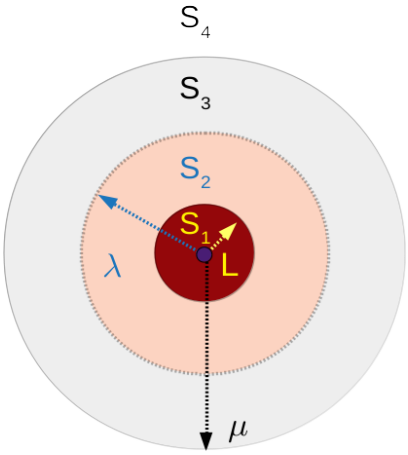


Figure B.1: Depiction of the sets that appear in Lemma 9

Lemma 9. *For non-negative weights $w_1, \dots, w_n$, vector $u \in \Delta_n$ and sets $S_1, \dots, S_4 \subseteq [n]$ as above it holds*

$$\begin{aligned}\sum_{i,j\in[n]} w_i^2\{u_i V_{ij} u_j\} \leq & \sum_{\ell\in[3],\ell'\in[3]} \sup_{\substack{i\in S_\ell,\\ j\in S_{\ell'}}} \left\{\frac{V_{ij}w_i}{w_j}\right\} \mu_\ell \mu_{\ell'} \\ & + u_{S_4} \sum_{\ell\in[3]} \sup_{\substack{i\in S_\ell,\\ j\in S_4}} \left\{V_{ij}\frac{w_i}{\mu}\right\} \mu_\ell \mu \\ & + \sup_{i\in S_4, j\in[n]} \{V_{ij}w_i\} \cdot \mu_4 \end{aligned} \tag{B.11}$$

where $u_S := \sum_{j\in S} u_j \leq 1$.

Proof of Lemma 4. First we observe that $S_1 \uplus S_2 \uplus S_3 \uplus S_4 = [n]$ forms a partition:

$$\begin{aligned}\sum_{i,j\in[n]} u_i u_j V_{ij} w_i^2 = & \sum_{\ell,\ell'\in[3]} \sum_{i\in S_\ell, j\in S_{\ell'}} u_i u_j V_{ij} w_i^2 \\ & + \sum_{\ell\in[3]} \sum_{i\in S_\ell, j\in S_4} u_i u_j V_{ij} w_i^2 \\ & + \sum_{i\in S_4, j\in[n]} u_i u_j V_{ij}.w_i^2 \end{aligned} \tag{B.12}$$

For the first three sets we have some bounds on the ration $\frac{w_i}{w_j}$ whereas for the last set we have a

bound on the w_i. We utilize these by:

$$\sum_{\substack{i\in S_\ell,\\ j\in S_{\ell'}}} \frac{V_{ij}w_i}{w_j} u_i w_i u_j w_j \le \sup_{\substack{i\in S_\ell,\\ j\in S_{\ell'}}} \{\frac{V_{ij}w_i}{w_j}\} \sum_{i\in S_\ell} w_i u_i \sum_{j\in S_{\ell'}} w_j u_j,$$

$$\sum_{\substack{i\in S_\ell,\\ j\in S_4}} \frac{V_{ij}w_i}{\mu} w_i u_i u_j \mu \le u_{S_4} \sup_{\substack{i\in S_\ell,\\ j\in S_4}} \left\{\frac{V_{ij}w_i}{\mu}\right\} \mu \sum_{i\in S_\ell} w_i u_i,$$

$$\sum_{\substack{i\in S_4,\\ j\in[n]}} \{V_{ij}w_i\} u_i u_j w_i \le \sup_{i\in S_4, j\in[n]} \{V_{ij}w_i\} \|u\|_1 \sum_{j\in S_4} w_i u_i.$$

Identifying μ_i in the above expressions and substituting the bounds in (B.12) completes the proof. □

B.1.3 Adaptive procedure

Theorem 8. *Given an (a,β,γ)-regular estimator $\mathcal{Z}$, the AMR procedure outputs a number $\hat{Z}$ such that*

$$\mathbb{P}[|\hat{Z}-\mu| \le \epsilon\cdot\max\{\mu,\tau\}] \ge \frac{2}{3} - O_{\gamma,\alpha}(\epsilon^2)$$

and with the same probability uses $O_\gamma(\frac{1}{\epsilon^2}\frac{1}{\mu^\beta})$ samples.

Proof of Theorem 8. Recall that $\mu_t = (1+\gamma)^{-t}$ and let $t_0 := t_0(\mu) \in \mathbb{Z}$ such that:

$$\mu_{t_0+1} \le \mu \le \mu_{t_0} \tag{B.13}$$

We consider two cases $t_0 < T$ or $t_0 \ge T$.

Case I ($t_0 < T$). In this case, we want to show that our algorithm with constant probability does not terminate before t_0 and not after t_0+1.

Let $\bar{Z}_t$ be the mean of m_t i.i.d. samples $Z_t^{(i)} \sim \mathcal{Z}(t,\gamma)$ with mean $\mathbb{E}[Z_t^{(i)}] = \mu$ and $\mathrm{RelVar}[Z_t^{(i)}] \le V_t(\mu)$. Then,

$$\mathrm{RelVar}[\bar{Z}_t] \le \frac{\epsilon^2}{6}\frac{V_t(\mu)}{V_t(\mu_{t+1})}. \tag{B.14}$$

Let A_0 be the event that the algorithm terminates before t_0.

$$\mathbb{P}[A_0] = \mathbb{P}[\exists t < t_0, \bar{Z}_t \geq \mu_t] \tag{B.15}$$

$$\leq \sum_{t<t_0} \mathbb{P}[\bar{Z}_t \geq \left(\frac{\mu_t}{\mu}\right)\mu] \tag{B.16}$$

$$\leq \frac{\epsilon^2}{6} \sum_{t=1}^{t_0-1} \frac{\mu^2}{(\mu_t-\mu)^2} \frac{V_t(\mu)}{V_t(\mu_{t+1})} \tag{B.17}$$

$$\leq \frac{\epsilon^2}{6} \sum_{t=1}^{t_0-1} \frac{\mu^2}{(\mu_t-\mu)^2} (\frac{\mu_{t+1}}{\mu})^{2-\alpha}. \tag{B.18}$$

where in (B.16) we use union bound, in (B.17) we use the first part of Lemma 6 and in (B.18) property (B) of a regular estimator. In the next three inequalities we use (B.13), $t \leq t_0 - 1$ and $\sum_{t=0}^{s} x^s \leq (1-x)^{-1}$ for $x < 1$.

$$\mathbb{P}[A_0] \leq \frac{\epsilon^2}{6} \sum_{t=1}^{t_0-1} \frac{1}{(1-\frac{\mu_{t_0}}{\mu_t})^2} \frac{\mu_{t+1}^2}{\mu_t^2} (\frac{\mu_{t_0}}{\mu_{t+1}})^{\alpha} \tag{B.19}$$

$$\leq \frac{\epsilon^2}{6} \frac{1}{\gamma^2} \mu_{t_0}^{\alpha} \sum_{t=1}^{t_0-1} (1+\gamma)^{-\alpha(t_0-t-1)} \tag{B.20}$$

$$\leq \frac{\epsilon^2}{6} \frac{1}{\gamma^2} \mu_{t_0}^{\alpha} \frac{1}{1-(1+\gamma)^{-\alpha}}. \tag{B.21}$$

Furthermore, let A_1 be the event that the algorithm terminates after $t > t_0 + 1$.

$$\mathbb{P}[A_1] = \mathbb{P}[\forall t \leq t_0 + 1, \bar{Z}_t < \mu_t] \tag{B.22}$$

$$\leq \mathbb{P}[\bar{Z}_{t_0+1} < \mu_{t_0+1}] \tag{B.23}$$

$$= 1 - \mathbb{P}[\bar{Z}_{t_0+1} \geq \mu_{t_0+1}]. \tag{B.24}$$

Using the second part of Lemma 6 (Paley-Zygmund)

$$\mathbb{P}[\bar{Z}_{t_0+1} \geq \mu_{t_0+1}] \geq \frac{1}{1 + \frac{(\gamma+1)^2}{\gamma^2} \frac{\epsilon^2}{6} \frac{V_{t_0+1}(\mu)}{V_{t_0+1}(\mu_{t_0+1})}} \tag{B.25}$$

$$\geq \left(1 + \frac{(\gamma+1)^2}{\gamma^2} \frac{\epsilon^2}{6}\right)^{-1}. \tag{B.26}$$

Therefore, $\mathbb{P}[A_1] \leq 1 - \left(1 + \frac{(\gamma+1)^2}{\gamma^2} \frac{\epsilon^2}{6}\right)^{-1} \leq \frac{(\gamma+1)^2}{\gamma^2} \frac{\epsilon^2}{6}$. Finally, let t^* be the (random) level where the algorithm terminates and A_2 be the event that $|\bar{Z}_{t^*} - \mu| > \epsilon\mu$. If any of the three events happen

we say that the procedure fails. We can bound the failure probability by:

$$\begin{aligned}\mathbb{P}[F] &= \mathbb{P}[A_0 \vee A_1 \vee A_2] \\ &= \mathbb{P}[A_0 \vee A_1 \vee A_2 \wedge \ A_0] + \mathbb{P}[(A_0 \vee A_1 \vee A_2) \wedge \ A_0^c] \\ &\leq \mathbb{P}[A_0] + \mathbb{P}[A_1 \wedge A_0^c] + \mathbb{P}[A_2 \wedge A_0^c]. \end{aligned} \tag{B.27}$$

To bound the last term we use:

$$\begin{aligned}\mathbb{P}[A_2 \wedge A_0^c] &= \mathbb{P}[A_2 \wedge A_0^c \wedge A_1] + \mathbb{P}[A_2 \wedge A_0^c \wedge A_1^c] \\ &\leq \mathbb{P}[A_1] + \mathbb{P}[A_2 \wedge A_0^c \wedge A_1^c].\end{aligned}$$

and

$$\begin{aligned}\mathbb{P}[A_2 \wedge A_0^c \wedge A_1^c] &= \sum_{t \in \{t_0, t_0+1\}} \mathbb{P}[|\bar{Z}_t - \mu| > \epsilon\mu \wedge t^* = t] \\ &\leq \sum_{t \in \{t_0, t_0+1\}} \mathbb{P}[|\bar{Z}_t - \mu| > \epsilon\mu] \\ &\leq \frac{1}{\epsilon^2} \sum_{t \in \{t_0, t_0+1\}} \text{RelVar}[\bar{Z}_t] \\ &\leq \frac{1}{\epsilon^2} \sum_{t \in \{t_0, t_0+1\}} \frac{\epsilon^2}{6} \frac{V_t(\mu)}{V_t(\mu_{t+1})}.\end{aligned}$$

By definition $\mu \geq \mu_{t+1}$ for all $t \geq t_0$, thus by (B) and (B.28):

$$\mathbb{P}[A_2 \wedge A_0^c \wedge A_1^c] \leq \frac{2}{6} = \frac{1}{3}. \tag{B.28}$$

Hence, the overall probability failure is bounded by:

$$\begin{aligned}\mathbb{P}[F] &\leq \mathbb{P}[A_0] + 2\mathbb{P}[A_1] + \mathbb{P}[A_2 \wedge A_0^c \wedge A_1^c] \\ &\leq \frac{\epsilon^2}{6} \frac{1}{\gamma^2} \mu_{t_0}^{\alpha} \frac{1}{1 - (1+\gamma)^{-\alpha}} + 2\frac{(\gamma+1)^2}{\gamma^2} \frac{\epsilon^2}{6} + \frac{1}{3}.\end{aligned}$$

When the algorithm succeeds the total number of samples is bounded by

$$\begin{aligned}\sum_{t=1}^{t_0+1}\lceil\frac{6}{\epsilon^2}V_t(\mu_{t+1})\rceil &\le (t_0+1)+\frac{6C}{\epsilon^2}\sum_{t=1}^{t_0+1}(1+\gamma)^{\beta(t+1)}\\ &\le (t_0+1)+\frac{6C}{\epsilon^2}(1+\gamma)^{2\beta}\frac{(1+\gamma)^{\beta t_0}}{\gamma}\\ &\le (t_0+1)+\frac{6C}{\epsilon^2}\frac{(1+\gamma)^{\beta}}{\gamma}\frac{1}{\mu^{\beta}}.\end{aligned}$$

Case II ($t_0 \ge T$). In this case $\mu \le \mu_T \le \frac{1}{1+\gamma}\epsilon\tau$. By the same arguments as in the case $t_0 < T$ we get that the probability terminates before $t < t_0$ is at most $\frac{\epsilon^2}{6\gamma^2}\mu_{t_0}^{\alpha}\frac{1}{1-(1+\gamma)^{-\alpha}}$. If the condition $\bar{Z}_T \ge \mu_T$ is satisfied then:

$$\mathbb{P}[|\bar{Z}_T-\mu|>\epsilon\mu] \le \frac{1}{\epsilon^2}\mathrm{RelVar}[\bar{Z}_T] \le \frac{1}{6} \tag{B.29}$$

If $\bar{Z}_T < \mu_T$ then:

$$|0-\mu| \le \mu \le \mu_T \le \frac{1}{1+\gamma}\epsilon\tau \le \epsilon\max\{\mu,\tau\} \tag{B.30}$$

Conclusion. Thus, overall if $\hat{Z}$ is the output of AMR:

$$\begin{aligned}\mathbb{P}[|\hat{Z}-\mu|>\epsilon\max\{\mu,\tau\}] \le{}& \frac{\epsilon^2}{6}\frac{1}{\gamma^2}\mu_{t_0}^{\alpha}\frac{1}{1-(1+\gamma)^{-\alpha}}\\ &+2\frac{(\gamma+1)^2}{\gamma^2}\frac{\epsilon^2}{6}+\frac{1}{3}\end{aligned}$$

As we see in the above expression the failure probability is dominated by the $\frac{1}{3}$ term. For example for $\gamma = 1, \epsilon = 0.2$, $\alpha = 1$ we have that the extra term is less than 0.0667. □

B.1.4 Regular estimator for Gaussian Kernel

Theorem 9. *$\mathcal{Z}_{\mathrm{Gauss}}$ is $(1, \frac{3}{4}, \gamma)$-regular and takes preprocessing time/space bounded by $O_{d,\kappa_T,\gamma}(\epsilon^{-3+\frac{1}{4}}\tau^{-\frac{3}{4}}\cdot n)$.*

Proof of Theorem 9. By Lemma 8 and Theorem 1 (Section 4.1), (A) holds with $V_t(\mu) := \frac{4e^{\frac{3}{2}}}{\mu}e^{r_t^2-r_t\sqrt{\log(\frac{1}{\mu})}}$. Moreover, since $\forall x \ge y > 0$

$$\frac{V_t(y)}{V_t(x)} = \frac{x}{y}e^{-r_t(\sqrt{\log(\frac{1}{y})}-\sqrt{\log(\frac{1}{x})})} \le \left(\frac{x}{y}\right)^{2-1} \tag{B.31}$$

and $V_t^{'}(x) = -\frac{4e^{\frac{3}{2}}}{x} e^{r_t^2 - r_t\sqrt{\log(\frac{1}{\mu})}}(\frac{1}{x} + \frac{r_t}{2\sqrt{\log(\frac{1}{x})}}) < 0$, property (B) holds with $\alpha = 1$. Finally,

$$V_t(\mu_{t+1}) = 4e^{\frac{3}{2}} e^{\{\frac{1}{4} - \frac{1}{2}\sqrt{\frac{t+1}{t}} + (1+\frac{1}{t})\} t\log(1+\gamma)} \tag{B.32}$$

$$= 4e^{\frac{3}{2}} \left(\frac{1}{\mu_t}\right)^{\frac{1}{4} - \frac{1}{2}\sqrt{\frac{t+1}{t}} + (1+\frac{1}{t})} \tag{B.33}$$

$$\leq 4e^{\frac{3}{2}} (1+\gamma)^{1-\frac{1}{\sqrt{2}}} \cdot \left(\frac{1}{\mu_t}\right)^{\frac{3}{4}}, \tag{B.34}$$

and consequently (C) holds with $\beta = \frac{3}{4}$. Finally, the estimator uses at most $O(\frac{1}{\epsilon^2} V_T(\mu_{T+1}))$ hash tables each taking preprocessing time/space $O_{d,q_T,\gamma}(n)$ space. □

B.2 Hashing-based Sketch

For any hash table H and a vector $u \in \Delta_n$ (simplex), let $B = B(H)$ denote the number of buckets and $u_{\max} = u_{\max}(H) := \max\{u_{H_i} : i \in [B]\}$ the maximum weight of any hash bucket of H. The precise definition of our Hashing-Based-Sketch is given in Algorithm 7.

For a fixed H, we can obtain the following bounds on the first two moments of our sketch (S_m, w).

Lemma 10 (Moments)**.** *For the sketch (S_m, w) produced by the HBS procedure it holds that*

$$\mathbb{E}[\mathrm{KDE}^w_{S_m}|H] = \mathrm{KDE}^u_P(q),$$
$$\mathrm{Var}[(\mathrm{KDE}^w_{S_m})^2|H] \leq \frac{1}{m}(Bu_{\max})^{1-\gamma^*} \sum_{i=1}^n k^2(x_i, q) u_i.$$

The above analysis shows that the sketch is always unbiased and that the variance depends on the hash function H only through $(Bu_{\max})^{1-\gamma^*} \geq 1$. We postpone the proof of this lemma after showing how it implies the following theorem.

Theorem 10. *Let H be the hash function sampled by the HBS procedure. For $\epsilon > 0$ and $\delta \in [e^{-\frac{6}{\epsilon^2}\frac{u_{\max}}{n\tau}}, e^{-\frac{6}{\epsilon^2}})$, let:*

$$\gamma^* = \left\{1 - \frac{\log(\frac{\epsilon^2}{6}\log(1/\delta))}{\log(\frac{u_{\max}}{\tau})}\right\}^{\mathbb{I}[B \leq (\frac{1}{2})^{\frac{1}{6}}\frac{1}{\tau}]}, \tag{B.35}$$

$$m = \frac{6}{\epsilon^2}\frac{1}{\tau}(Bu_{\max})^{1-\gamma^*} < \frac{\log(\frac{1}{\delta})}{\tau}. \tag{B.36}$$

Then (S_m, w) is an $(\epsilon, \frac{1}{6}, \tau)$-sketch and if $B \leq (\frac{1}{2})^{\frac{1}{6}}\frac{1}{\tau}$ any hash bucket with weight at least τ will have non empty intersection with S_m with probability at least $1 - \delta$.

Proof of Theorem 10. Given a hash bucket with weight at least τ, the probability that we sample a point from that bucket is at least:

$$\rho \geq \frac{\tau^\gamma}{B^{1-\gamma}} = \tau \frac{1}{(B\tau)^{1-\gamma}} \tag{B.37}$$

The probability that we see no point after m independent samples is less than $(1-\rho)^m \leq e^{-m\frac{\tau^\gamma}{B^{1-\gamma}}}$ For $m \geq \frac{\log(1/\delta)}{\tau}(B\tau)^{1-\gamma}$ this probability is at most δ. On the other hand by Lemma 10 if $m \geq \frac{6}{\epsilon^2}\frac{1}{\tau}\left(Bu_{\max}\right)^{1-\gamma}$ we have that $\mathrm{Var}[\mathrm{KDE}^w_{S_m}] \leq \frac{\epsilon^2}{6}\mu\tau$. The case $B > 2^{-\frac{1}{6}}\frac{1}{\tau}$ is trivial as $\gamma^* = 1$. For $B \leq 2^{-\frac{1}{6}}\frac{1}{\tau} \Rightarrow u_{\max} \geq \frac{1}{B} \geq \tau 2^{\frac{1}{6}}$. We set γ to make the two lower bounds on m equal,

$$\frac{6}{\epsilon^2}\frac{1}{\tau}\left(Bu_{\max}\right)^{1-\gamma} = \frac{\log(1/\delta)}{\tau}(B\tau)^{1-\gamma} \tag{B.38}$$

$$\Leftrightarrow \left(\frac{u_{\max}}{\tau}\right)^{1-\gamma} = \frac{\epsilon^2\log(1/\delta)}{6} \tag{B.39}$$

$$\Leftrightarrow \gamma = 1 - \frac{\log(\frac{\epsilon^2}{6}\log(1/\delta))}{\log(\frac{u_{\max}}{\tau})}. \tag{B.40}$$

This is strictly less than one for $\log(1/\delta)\frac{\epsilon^2}{6} > 1 \Rightarrow \delta < e^{-\frac{6}{\epsilon^2}}$, and more than zero for $\delta \geq e^{-\frac{6}{\epsilon^2}\frac{u_{\max}}{\tau}}$. Since $u_{\max} \geq \tau 2^{1/6}$ the two inequalities are consistent. Furthermore,

$$m = \frac{6}{\tau\epsilon^2}\cdot\left(Bu_{\max}\right)^{1-\gamma^*} \tag{B.41}$$

$$= \frac{6}{\tau\epsilon^2}\cdot\left(Bu_{\max}\right)^{\log(\frac{\epsilon^2\log(1/\delta)}{6})\frac{1}{\log(\frac{u_{\max}}{\tau})}} \tag{B.42}$$

$$= \frac{6}{\tau\epsilon^2}\cdot e^{\log(\log(1/\delta)\frac{\epsilon^2}{6})\frac{\log(Bu_{\max})}{\log(\frac{u_{\max}}{\tau})}} \tag{B.43}$$

$$\leq \frac{6}{\tau\epsilon^2}\cdot\left(\log(1/\delta)\frac{\epsilon^2}{6}\right)^{(1-\frac{1}{6}\frac{\log 2}{\log(\frac{u_{\max}}{\tau})})} \tag{B.44}$$

$$< \frac{\log(1/\delta)}{\tau}. \tag{B.45}$$

□

Remark 6. *Observe that* $\frac{\log\frac{1}{\delta}}{\tau}$ *is the number of samples that random sampling would require in order to have the same property for any bucket with* $u_{H_i} \geq \tau$*. When* $\gamma^* < 1$*, our scheme always uses less samples by a factor of* $\left(\log(1/\delta)\frac{\epsilon^2}{6}\right)^{\frac{\log(B\tau)}{\log(\frac{u_{\max}}{\tau})}} < 1$.

Thus, our sketch will have similar variance with random sampling in dense regions of the space but will have better performance for relatively "sparse" regions.

Proof of Lemma 10. Let I be the random hash bucket and X_I the corresponding random point,

then for a single point:

$$\begin{aligned}
\mathbb{E}[\mathrm{KDE}^{w_1}_{\{X_I\}}] &= \mathbb{E}_I[\mathbb{E}_{X_I}[\frac{u_{H_I}}{m}\frac{\sum_{i'=1}^{B} u^{\gamma}_{H_{i'}}}{u^{\gamma}_{H_I}}k(X_I,q)]] \\
&= \mathbb{E}_I[\sum_{j\in H_I}\frac{u_{H_I}}{m}\frac{\sum_{i'=1}^{B} u^{\gamma}_{H_{i'}}}{u^{\gamma}_{H_I}}k(x_j,q)\frac{u_j}{u_{H_I}}] \\
&= \frac{1}{m}\mathbb{E}_I[\frac{\sum_{i'=1}^{B} u^{\gamma}_{H_{i'}}}{u^{\gamma}_{H_I}}\sum_{j\in H_I}k(x_j,q)u_j] \\
&= \frac{1}{m}\sum_{i\in[B]}\sum_{j\in H_i}k(x_j,q)u_j \\
&= \frac{1}{m}\mathrm{KDF}^{u}_{P}(q).
\end{aligned}$$

The first part follows by linearity of expectation. Similarly,

$$\mathbb{E}[(\mathrm{KDF}^{w}_{S_m})^2] \le \sum_{j=1}^{m}\mathbb{E}[(\mathrm{KDF}^{w_j}_{\{x_j\}})^2] + (\mathrm{KDF}^{u}_{P}(q))^2.$$

By linearity we only have to bound the first term

$$\begin{aligned}
\mathbb{E}[(\mathrm{KDE}^{w_1}_{\{X_I\}})^2] &= \mathbb{E}_I[\mathbb{E}_{X_I}[(\frac{u_{H_I}}{m}\frac{\sum_{i'=1}^{B} u^{\gamma}_{H_{i'}}}{u^{\gamma}_{H_I}}k(X_I,q))^2]] \\
&= \mathbb{E}_I[\sum_{j\in H_I}(\frac{u_{H_I}}{m}\frac{\sum_{i'=1}^{B} u^{\gamma}_{H_{i'}}}{u^{\gamma}_{H_I}}k(x_j,q))^2\frac{u_j}{u_{H_I}}] \\
&= \mathbb{E}_I[(\frac{\sum_{i'=1}^{B} u^{\gamma}_{H_{i'}}}{mu^{\gamma}_{H_I}})^2 u_{H_I}\sum_{j\in H_I}k^2(x_j,q)u_j] \\
&= \frac{\sum_{i'=1}^{B} u^{\gamma}_{H_{i'}}}{m^2}\sum_{i\in[B]}u^{1-\gamma}_{H_I}\sum_{j\in H_I}k^2(x_j,q)u_j \\
&\le \frac{\sum_{i'=1}^{B} u^{\gamma}_{H_{i'}}}{m^2}u^{1-\gamma}_{\max}\sum_{i\in[B]}\sum_{j\in H_I}k^2(x_j,q)u_j \\
&\le \frac{(Bu_{\max})^{1-\gamma}}{m^2}\sum_{j=1}^{n}k^2(x_j,q)u_j.
\end{aligned}$$

The last inequality follows by applying Hölder's inequality with $p = \frac{1}{\gamma}$ and $q = \frac{1}{1-\gamma}$, and due to $u \in \Delta_n$. □

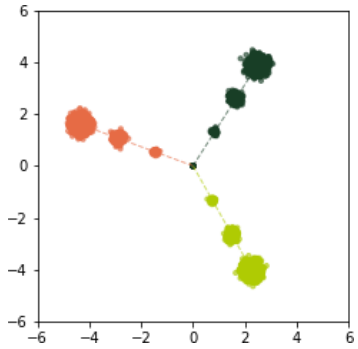


Figure B.2: ($\mu = 0.01$, $D = 3$, $s = 4$, $d = 2$, $\sigma = 0.05$)-Instance. Each of the $D = 3$ directions is coded with a different color.

B.3 Synthetic Benchmarks

In this section, we introduce a general procedure to create tunable synthetic datasets that exhibit different local structure around the query. We then show how to use this procedure as a building block to create two different families of instances with specific characteristics aimed to test kernel density evaluation methods.

$(\mu, D, n, s, d, \sigma)$-Instance. Since the problem of kernel density is query dependent and the kernel typically depends only on the distance, we shall always assume that the query point is at the origin $q = 0 \in \mathbb{R}^d$.

We further assume that the kernel is an invertible function of the distance $K(r) \in [0, 1]$ and let $K^{-1}(\mu) \in [0, \infty)$ be the inverse function. For example, the exponential kernel is given by $K(r) = e^{-r}$ and the inverse function is given by $K^{-1}(\mu) = \log(\frac{1}{\mu})$.

The dataset is created with points lying in D different directions and s distance scales (equally spaced between 0 and $R = K^{-1}(\mu)$) such that the *contribution from each direction and scale* to the kernel density at the origin is *equal*. To achieve this the number of points n_j placed a at the j-th distance scale r_j is given by

$$n_\ell := \lfloor n \frac{\mu}{K(r_j)} \rfloor. \tag{B.46}$$

The reasoning behind this design choice is to make sure that we have diversity in the distance scales that matter in the problem, so not to favor a particular class of methods (e.g. random sampling , nearest-neighbor based). Also, placing the points on the same direction makes the instance more difficult for HBE as the variance in (4.4) increases with the ratio $\frac{\mathbb{P}[h(i)=h(j)=h(q)]}{\mathbb{P}[h(i)=h(q)]^2}$, that expresses how correlated the values $\{h(i), h(j), h(q)\}$ are. We give an example visualization of such data sets in 2 dimensions in Figure B.2. The detailed procedure is described below (Algorithm 15).

Remark 7. *If $D \ll n$ this class of instances becomes highly structured with a small number of tightly knit "clusters" (Figure B.2). One would expect in this case, space-partioning methods to perform*

Algorithm 15 $(\mu, D, n, s, d, \sigma)$-Instance

Input: $\mu \in [\frac{1}{n}, 1]$, $D \geq 1$, $n \geq 1$, $s \geq 2$, $d \geq 1$, $\sigma \geq 0$, kernel K, inverse K^{-1}.
$R \leftarrow K^{-1}(\mu)$, $r_0 \leftarrow K^{-1}(1)$, $P \leftarrow \emptyset$.
for $j = 0, \ldots, s-1$ **do**
    $r_{j+1} \leftarrow \frac{R-r_0}{s-1} j + r_0$ ▷ distances for each D
    $n_{j+1} \leftarrow \lfloor n \frac{\mu}{K(r_{j+1})} \rfloor$ ▷ points at each distance
end for
for $i = 1, \ldots, D$ **do**
    $v_i \leftarrow \frac{g_i}{\|g_i\|}$ with $g_i \sim \mathcal{N}(0, I_d)$. ▷ random direction
    for j=1,. . . , s **do** ▷ For each distance scale
        for $\ell = 1, \ldots, n_j$ **do** ▷ generate a "cluster"
            $g_{ij\ell} \sim \mathcal{N}(0, I_d)$
            $x_{ij\ell} \leftarrow s_j v_i + \frac{\sigma}{\sqrt{d}} s_j g_{ij\ell}$
            $P \leftarrow P \cup \{x_{ij\ell}\}$
        end for
    end for
end for
Output: Set of points P

well. At the same time by Lemma 4, these instances are the ones that maximize the variance of both HBE $(s \geq 1)$ *and RS* $(s > 1)$ *.*

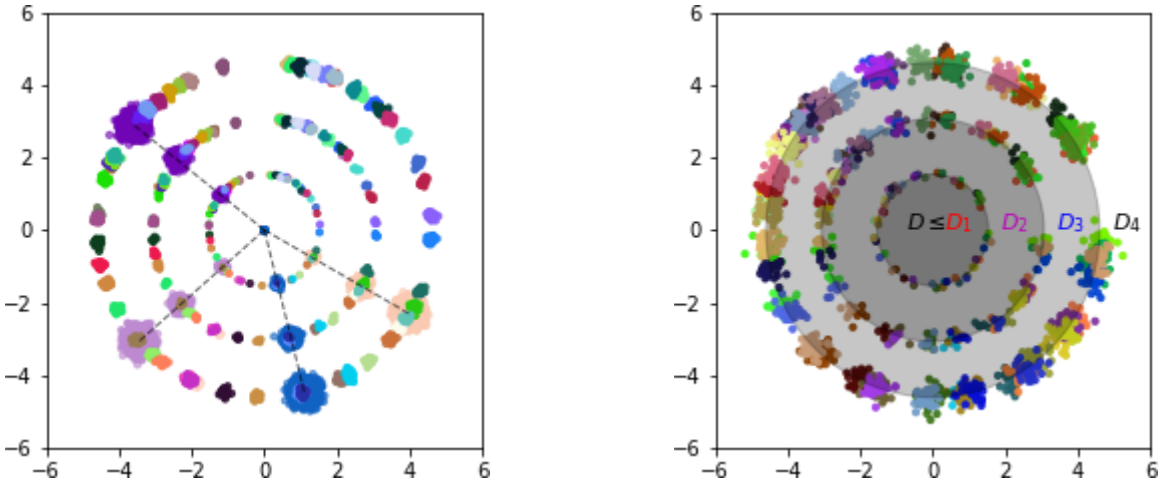
Remark 8. *On the other hand if* $D \gg n$ *the instances become spread out (especially in high dimensions). This type of instances is ideal for sampling based methods when* $s = 1$*, and difficult for space-partitioning methods.*

Based on the above remarks we propose the following sub-class of instances.

"Worst-case" instance. In order to create an instance that is *hard for all methods* we take a union of the two extremes $D \ll n$ and $D \gg n$. We call such instances "worst-case" as there does not seem to be a single type of structure that one can exploit, and these instances realize the worst-case variance bounds for both HBE and RS. In particular, if we want to generate an instance with N points, we first set D a small constant and $n = \Theta(N)$ and take the union of such a dataset with another using $D = \Omega(N^{1-o(1)})$ and $n = O(N^{o(1)})$. An example of such a dataset is given in B.3a.

D-structured instance. "Worst-case" instances are aimed to be difficult for any kernel evaluation method. In order to create instances that have more varied structure, we use our basic method to create a single parameter family of instances by fixing N, μ, σ, s, d and setting $n = \frac{N}{D}$. We call this family of instances as *D-structured*. As one increases D, two things happen:

- The number of directions (clusters) increases.
- $n = \frac{N}{D}$ decreases and hence certain distance scales disappear. By (B.46), if $n\mu < K(r_j) \Rightarrow D_j > \frac{N\mu}{K(r_j)}$ then distance scale j will have no points assigned to it.



(a) "worst-case" instance (b) difficult case

Figure B.3: The two families of instances for $d = 2$.

Hence, for this family when $D \ll \frac{N}{D} \leftrightarrow D \ll \sqrt{N}$ the instances are highly structured and we expect space-partitioning methods to perform well. On the other extreme as D increases and different distance scales start to die out (Figure B.3b) the performance of random sampling keeps improving until there is only one (the outer) distance scale, where random sampling will be extremely efficient. On the other hand HBE's will have roughly similar performance on the two extremes as both correspond to worst-case datasets for scale-free estimators with $\beta = 1/2$, and will show slight improvement in between $1 \ll D \ll n$. This picture is confirmed by our experiments.

B.4 Additional Results

B.4.1 Datasets

We provide detailed descriptions of the datasets as well as the bandwidth used for the kernel density evaluation in Table B.2. We also include specifications for the additional datasets acquired from LIBSVM [61] and the UCI Machine Learning Repository [206] that were used to evaluate the accuracy of the diagnostic procedure.

We selected the top eight datasets in Table B.2 for density evaluation in the main paper as they are the largest, most complex datasets in our collection. We provide additional density evaluation results in Table B.1 for datasets with comparable sizes or dimensions to the ones reported in the main paper. For higgs and hep, FigTree failed to finish the evaluation within a day. Given the performance of RS on these datasets, we don't expect FigTree to achieve better performance even if the query returns successfully. For mnist, we were not able to get ASKIT to achieve relative error below 1 even after trying parameters that span a few orders of magnitude; this is potentially caused

Table B.1: Precomputatation time (*init*) and total query time (*query*) on 10K random queries for additional datasets. All runtime measurements are reported in seconds.

Dataset	Time	RS	HBE	ASKIT	FigTree
higgs	*init*	0	141	25505	> 1day
	query	**6**	18	1966	> 1day
hep	*init*	0	138	23421	> 20 hours
	query	**6**	11	1581	> 1day
susy	*init*	0	67	5326	3245
	query	18	**12**	> 9756	5392
home	*init*	0	11	237	7
	query	2369	**17**	376	33
mnist	*init*	0	211	14	437
	query	**168**	389	?	1823

by the high-dimensionality and sparsity of this dataset.

B.4.2 Synthetic Experiment

For the clustering test, we set $\mu = 0.001, s = 4, d = 100, \sigma = 0.01, N = 500K$. The varying parameters are the number of clusters (D) and the number of points per cluster n. We report precomputatation time (in seconds) for all methods in Table B.3. The ordering of methods according to precomputatation time largely follows that of query time.

As discussed in Section B.3, for the D-structured instances as the number of points per cluster n decreases, smaller distance scales start to disappear due to (B.46). Let D_i be the threshold such that for $D > D_i$, there are no-points in scale i. The corresponding numbers for our experiment is roughly $D1 = 500$ (at distance 0) , $D2 = 1077$, $D3 = 10K$, $D4 = N = 500K$. In particular, only a single distance scale remains when $D = 100K > D3$, a set up in which RS is orders of magnitude more efficient than alternative methods (Figure B.3a right).

B.4.3 Visualizations of real-world data sets

Our Log-Condition plots use circles with radius r to represent points with weights roughly e^{-r} (roughly at distance $\sqrt{r}$ for the Gaussian kernel). The visualizations are generated by plotting overlapping annuli around the origin that represent a random queries from the dataset, such that the width of the annulus roughly corresponds to the log of the relative variance of random sampling.

We observe two distinctive types of visualizations. Datasets like census exhibit dense inner circles, meaning that a small number of points close to the query contribute significantly towards the density. To estimate the density accurately, one must sample from these small clusters, which HBE does better than RS. In contrast, datasets like MSD exhibit more weight on the outer circles,

Table B.2: Specifications of real-world datasets.

Dataset	N	d	σ	Description
census	$2.5M$	68	3.46	Samples from 1900 US census.
TMY3	$1.8M$	8	0.43	Hourly energy load profiles for reference buildings.
TIMIT	$1M$	440	10.97	Speech data for acoustic-phonetic studies. First 1M data points used.
SVHN	$630K$	3072	28.16	Google Street View house numbers. Raw pixel values of 32x32 images.
covertype	$581K$	54	2.25	Cartographic variables for predicting forest cover type.
MSD	$463K$	90	4.92	Audio features of popular songs.
GloVe	$400K$	100	4.99	Pre-trained word vectors from Wikipedia 2014 + Giga 5 word. 5. 6B tokens, 400K vocab.
ALOI	$108K$	128	3.89	Color image collection of 1000 small objects. Each image is represented by a 128 dimensional SIFT vector.
higgs	$11M$	28	3.41	Signatures of Higgs bosons from Monte Carlo simulations.
hep	$10.5M$	27	3.36	Signatures of high energy physics particles.
susy	$5M$	18	2.24	Signatures of supersymmetric particles.
home	$969K$	10	0.53	Home gas sensor measurements.
skin	$245K$	3	0.24	Skin Segmentation dataset.
ijcnn	$142K$	22	0.90	IJCNN 2001 Neural Network Competition.
acoustic	$79K$	50	1.15	Vehicle classification in distributed sensor networks.
mnist	$70K$	784	11.15	28x28 images of handwritten digits.
corel	$68K$	32	1.04	Image dataset, with color histograms as features.
sensorless	$59K$	48	2.29	Dataset for sensorless drive diagnosis.
codrna	$59K$	8	1.13	Detection of non-coding RNAs.
shuttle	$44K$	9	0.62	Space shuttle flight sensors.
poker	$25K$	10	1.37	Poker hand dataset.
cadata	$21K$	8	0.62	California housing prices.

meaning that a large number of "far" points is the main source of density. Random sampling has a good chance of seeing these "far" points, and therefore, tends to perform better on such datasets. The top plots in Figure B.4 amplify these observations on synthetic datasets with highly clustered/scattered structures. RS performs better for all datasets in the second and fourth columns except for SVHN.

Table B.3: Precomputatation time (in seconds) for clustering test.

n	D	HBE	FigTree	ASKIT
$500K$	1	192	2	113
$50K$	10	20	3	105
$5K$	100	16	16	105
500	1000	19	174	104
50	10000	39	1516	102
5	100000	334	0.3	101

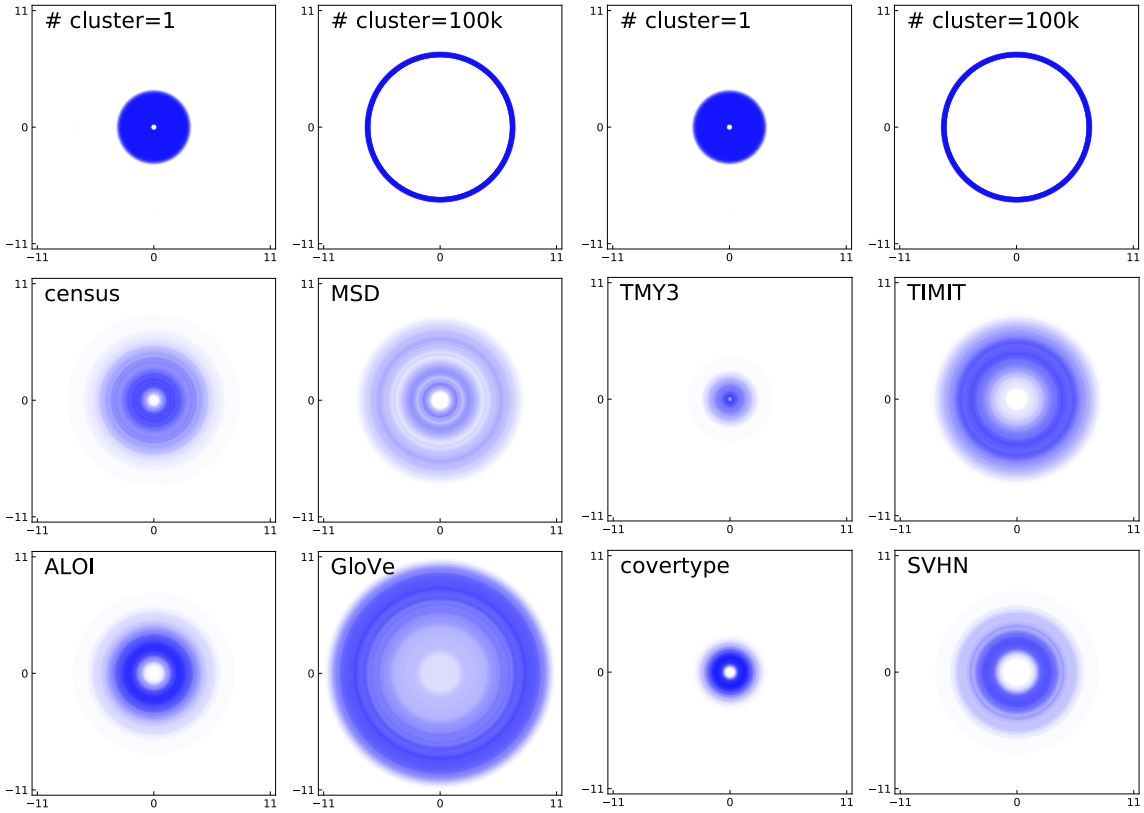


Figure B.4: Visualizations of datasets. The top row shows two extreme cases of highly clustered (# cluster=1) versus highly scattered (#cluster=100k) datasets. RS performs better for all datasets in the second and fourth columns except for SVHN.

Appendix C

Supplementary material for ASAP

C.1 Analysis

C.1.1 Roughness Estimate

We start with a detailed derivation for Equation 5.5 in Section 5.2.2. Given the original time series $X : \{x_1, x_2, ..., x_N\}$ (a weakly stationary process), and the smoothed series $Y : \{y_1, y_2, ..., y_{N-w}\}$ obtained by applying a moving average of window size w, we want to show that:

$$\text{roughness}(Y) = \frac{\sqrt{2}\sigma}{w}\sqrt{1 - \frac{N}{N-w}\text{ACF}(X, w)}$$

Note that in Equation 5.1, when the IID assumption does not hold, $\text{cov}(X_f, X_l) \neq 0$. The covariance of discrete two random variables X, Y each with a set of N equal-probability values is defined as:

$$\text{cov}(X, Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mathbb{E}(X))(y_i - \mathbb{E}(Y))$$

And for a discrete process, given N equi-spaced observations of the process $x_1, x_2, ..., x_N$, an estimate of the autocorrelation function at lag k can be obtained by:

$$\text{ACF}(X, w) = \frac{\sum_{i=1}^{N-w}(x_i - \bar{x})(x_{i+w} - \bar{x})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Therefore, we can rewrite the autocorrelation function as:

$$\text{ACF}(X, w) = \frac{(N-w)\text{cov}(X_f, X_l)}{N\sigma^2}, \text{ or } \text{cov}(X_f, X_l) = \frac{N\sigma^2}{N-w}\text{ACF}(X, w)$$

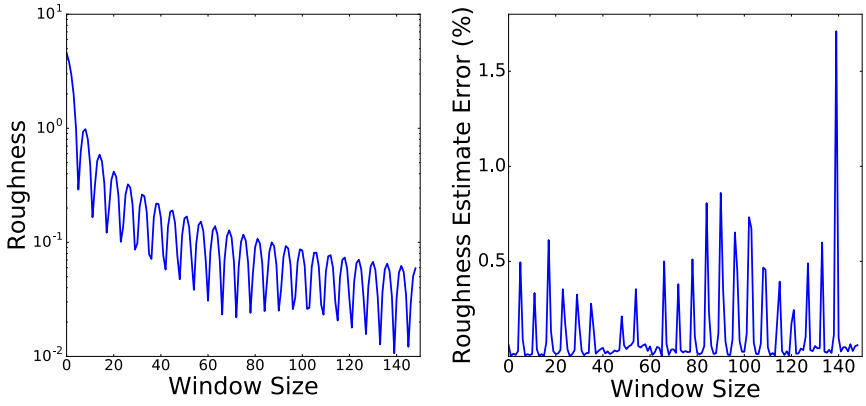


Figure C.1: True roughness and percent error of roughness estimation (Equation 5.5) over window sizes for dataset Temp. Estimate errors are within 1.2% of the true value across all window sizes.

Substituting $\text{cov}(X_f, X_l)$ into (5.1), we obtain:

$$\begin{aligned}\text{roughness}(Y) &= \frac{1}{w}\sqrt{\sigma^2 + \sigma^2 - 2\frac{N\sigma^2}{N-w}\text{ACF}(X,w)} \\ &= \frac{\sqrt{2}\sigma}{w}\sqrt{1 - \frac{N}{N-w}\text{ACF}(X,w)}\end{aligned}$$

We empirically evaluate the accuracy of the roughness estimation (Equation 5.5) on the Temp dataset, and report the relative error in percent (Figure C.1). For this time series, the roughness of the aggregated series drops sharply at window sizes around multiples of 6, which correspond to the autocorrelation peaks. Furthermore, estimated roughness (via Equation 5.5) is within 1.2% of the true value across all window sizes.

C.1.2 Impact of Pixel-aware Preaggregation

We first provide an analysis for the pixel-aware preaggregation strategy. Given a time series of length N sampled from uniform distribution and a target resolution of t pixels, we have a point-to-pixel ratio of $p_a = \frac{N}{t}$. Let w_{opt} be the window size that minimizes the roughness on the original time series. Note that searching on preaggregated data is equivalent to only selecting window sizes that are multiples of p_a. Since roughness decreases and kurtosis increases with window size, the optimal window size over the preaggregated data is $w_a = \lfloor \frac{w_{opt}}{p_a} \rfloor$, or $w_a p_a \leq w_{opt} < (w_a + 1)p_a$. Therefore, $\frac{w_{opt}}{w_a p_a} < \frac{(w_a+1)p_a}{w_a p_a} = \frac{w_a+1}{w_a}$. Recall roughness scales proportionally with $\frac{1}{w}$ (Equation 5.2), so preaggregation incurs a penalty of no more than $\frac{w_a+1}{w_a}$ in roughness. Intuitively, as optimal window size increases, quality of preaggregation increases and in the limit, recovers the same solution as the search over the original data. We have a similar analysis for periodic data (roughness varies with $\frac{1}{w}\sqrt{1 - ACF(w)}$). Let the autocorrelation corresponding to w_{opt} be ACF_{opt}, and let the maximum

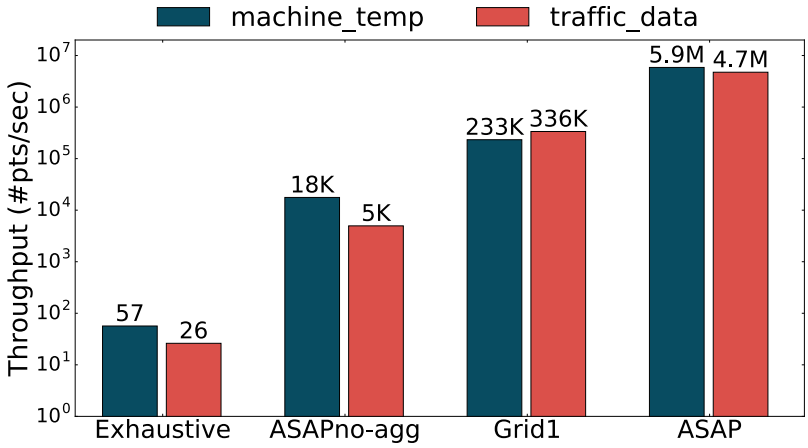


Figure C.2: Throughput of exhaustive search and ASAP on two datasets (machine_temp, traffic_data), without and with pixel-aware preaggregation for a target resolution of 1200 pixels. ASAP on preaggreaged data is up to 5 order of magnitude faster than exhaustive search on raw data.

Name	**Description**
Exhaustive	Exhaustive search on raw time series
ASAPno-agg	ASAP on raw time series
Grid1	Exhaustive search on preaggregated data
Grid2	Exhaustive search with step size 2 on preaggregated data
Grid10	Exhaustive search with step size 10 on preaggregated data
Binary	Binary search on preaggregated data
ASAP	ASAP on preaggregated data

Table C.1: Descriptions of search strategies used in performance evaluations.

change in autocorrelation along a window of size p_a be ACF_Δ. Specifically, while w_{opt} may be able to pick an autocorrelation peak (i.e., a window size with high autocorrelation), searching on preaggregated data may only come within p_a of the peak. By examining the maximum rise of the autocorrelation function over a period of length p_a, we can bound the impact of roughness as above by $\frac{w_a+1}{w_a}\sqrt{\frac{1-ACF_{opt}+ACF_\Delta}{1-ACF_{opt}}}$. This implies that the impact of preaggregation on periodic data depends on the sharpness of the autocorrelation function, which is in turn dataset-dependent. Our empirical results confirm that both of these effects are limited on real-world datasets.

In addition, we evaluate the preaggregation strategy's impact on performance. Figure C.2 shows the throughput of running exhaustive search and ASAP on two similar sized datasets (machine_temp and traffic_data), before and after applying the pixel-aware preaggregation. Table C.1 provides details of the search strategies used in the evaluation. With a target resolution of 1200 pixels, ASAP on aggregated series is up to 5 order of magnitude faster compared to an exhaustive search on the original time series.

C.2 Additional Results

C.2.1 Alternative Smoothing Functions

In ASAP, we deliberately choose simple moving average (SMA) as the smoothing function due to its wide usage in monitoring dashboards. However, there are many smoothing functions for time series data in the broader signal processing design space. Here, we perform a qualitative comparison of moving average (SMA) with alternative smoothing functions including Fast Fourier transform, Savitzky-Golay filter and the minmax aggregation. Specifically, we compare the achieved roughness of different smoothing functions using the same parameter selection criteria (minimizing roughness subject to kurtosis preservation).

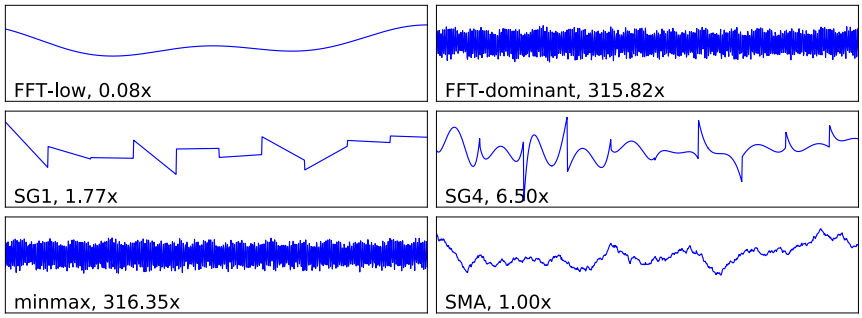
We varied the window sizes for Savitzky-Golay and minmax filters, and varied the number frequency components included in the reconstruction for FFTs. Specifically, SG1 approximates data points in a window using a line while SG4 approximates using a polynomial of degree 4; FFT-low reconstructs the signal by composing components in the order of increasing frequency while FFT-dominant composes frequency components of decreasing power. Figure C.3 reports the achieved roughness compared to SMA for each smoothing function and each dataset used in the user study. Overall, we found that FFT-dominant and minmax result in high roughness: the former tend to keep the dominant high frequencies in the original time series during reconstruction and the latter, by definition, produces smoothed time series where consecutive points are maximized in distance in the given window. FFT-low, SG1 and SG4, on the other hand, produce smoother plots and occasionally outperform SMA in roughness. Figure C.3 presents all smoothed plots for visual comparison.

C.2.2 Sample Visualizations

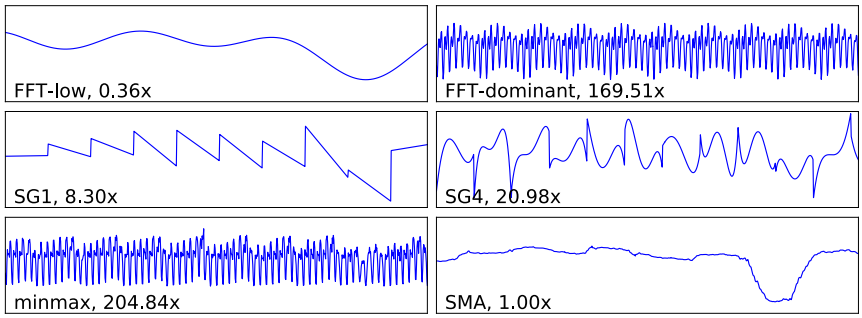
Figure C.4 presents a subset of visualizations of raw and (ASAP-)smoothed time series for datasets described in Table 5.2. Both exhaustive search and ASAP left the Twitter_AAPL dataset unsmoothed due to its high starting kurtosis (Figure C.4f).

Figure C.5 presents visualizations of the Temp dataset plotted using four popular monitoring systems and plotting libraries: Excel, Prometheus, Tableau and Grafana. None of the above tools automatically smooth time series to suppress noise.

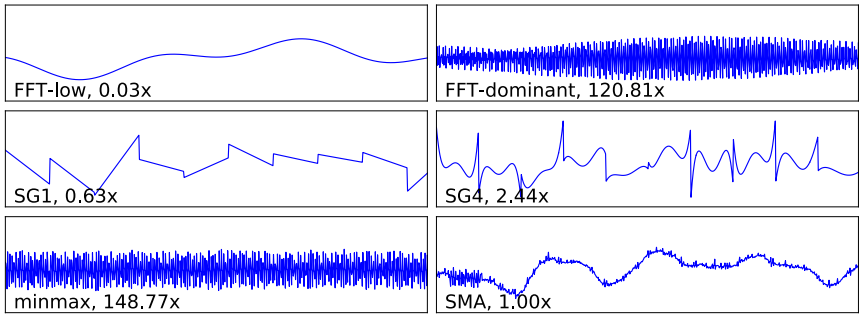
...



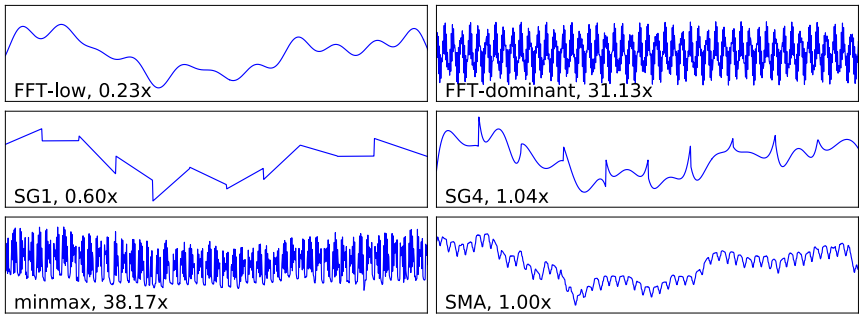
(a) Temp



(b) Taxi

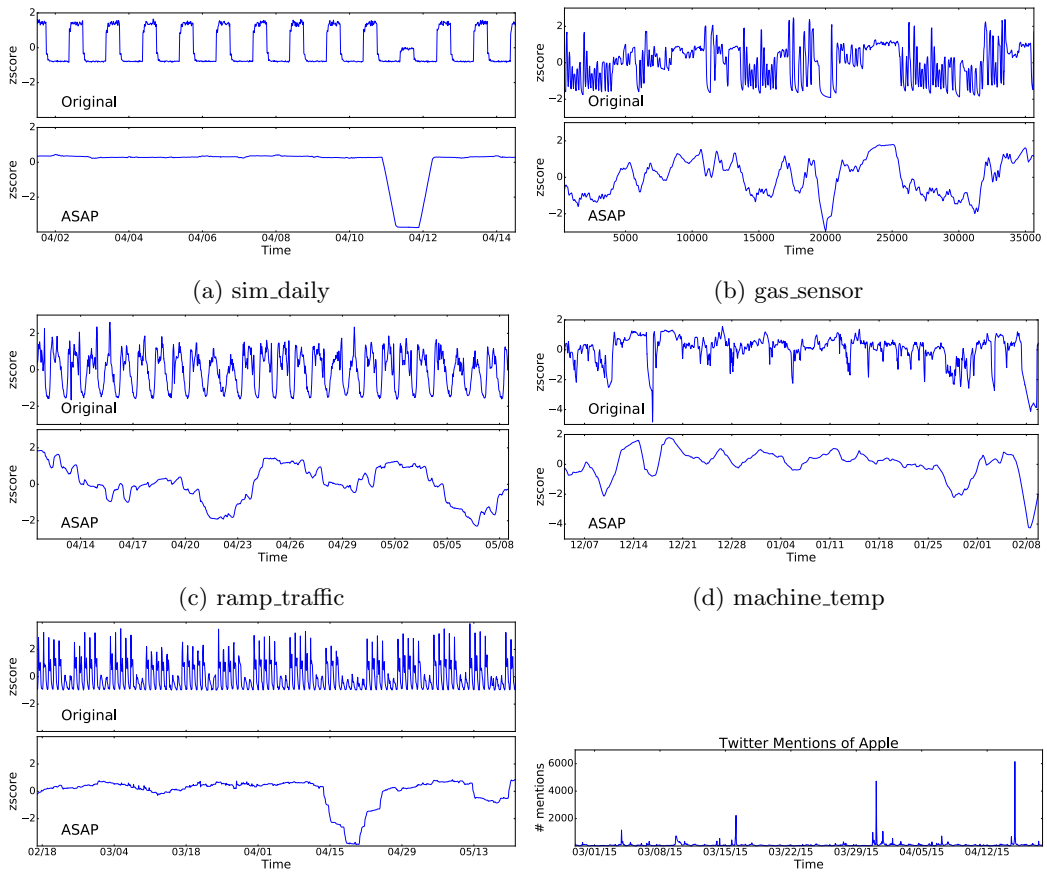


(c) EEG



(d) Power

Figure C.3: Achieved roughness of FFT, Savitzky-Golay filter and Wiener filter over SMA.

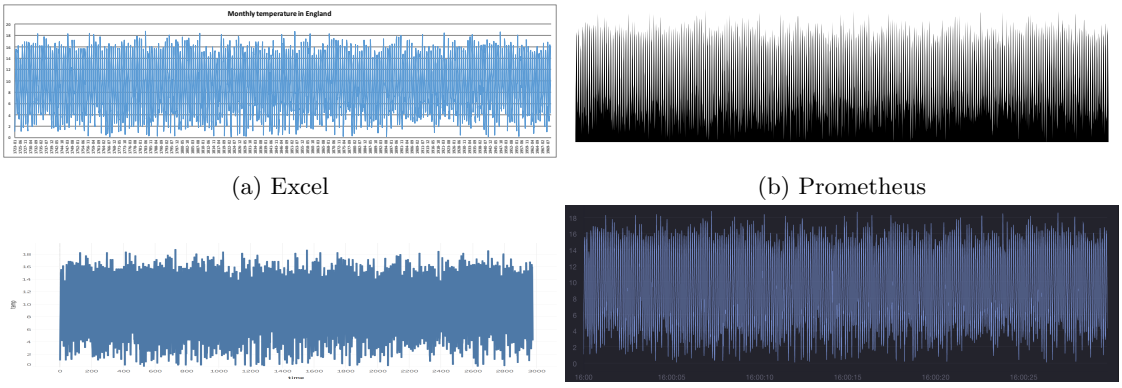


(a) sim_daily (b) gas_sensor

(c) ramp_traffic (d) machine_temp

(e) traffic_data (f) Twitter_AAPL

Figure C.4: Original and ASAP-smoothed plots. The twitter dataset is left unsmoothed by both exhaustive search and ASAP due to its high initial kurtosis.



(a) Excel (b) Prometheus

(c) Tableau (d) Grafana

Figure C.5: Sample visualizations for the Temp dataset using various existing time series visualization tools; none automatically smoothes out the noise.

Bibliography

[1] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, et al. Diff: a relational interface for large-scale data explanation. *PVLDB*, 12(4):419–432, 2018.

[2] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, et al. Diff: a relational interface for large-scale data explanation. *The VLDB Journal*, 30(1):45–70, 2021.

[3] Swarup Acharya, Phillip B Gibbons, and Viswanath Poosala. Aqua: A fast decision support system using approximate query answers. *PVLDB*, 1999.

[4] Swarup Acharya, Phillip B Gibbons, and Viswanath Poosala. Congressional samples for approximate answering of group-by queries. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 487–498, 2000.

[5] Swarup Acharya, Phillip B Gibbons, and Viswanath Poosala. Congressional samples for approximate answering of group-by queries. In *SIGMOD*, pages 487–498, 2000.

[6] Colin Adams, Luis Alonso, Benjamin Atkin, John Banning, Sumeer Bhola, Rick Buskens, Ming Chen, Xi Chen, Yoo Chung, Qin Jia, Nick Sakharov, George Talbot, Adam Tart, and Nick Taylor. Monarch: Google's planet-scale in-memory time series database. *PVLDB*, 13(12):3181–3194, August 2020.

[7] Sameer Agarwal, Henry Milner, Ariel Kleiner, Ameet Talwalkar, Michael Jordan, Samuel Madden, Barzan Mozafari, and Ion Stoica. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 481–492, 2014.

[8] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42, 2013.

[9] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[10] Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawhney, and Kyuseok Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*, pages 490–501, 1995.

[11] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of time-oriented data*. Springer, 2011.

[12] Muhammad Intizar Ali et al. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets. In *ISWC*, pages 374–389, 2015.

[13] SCEDC (2013): Southern California Earthquake Center. Caltech. Dataset. doi:10.7909/C3WD3xH1.

[14] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[15] Amazon S3 pricing. `https://aws.amazon.com/s3/pricing/`. Accessed July 23, 2021.

[16] Don L Anderson. *Theory of the Earth*. Blackwell scientific publications, 1989.

[17] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[18] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and Optimal LSH for Angular Distance. *NIPS*, 1:1225–1233, 2015.

[19] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1018–1028. SIAM, 2014.

[20] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801, 2015.

[21] Arvind Arasu and Jennifer Widom. Resource sharing in continuous sliding-window aggregates. In *VLDB*, pages 336–347, 2004.

[22] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. Spark sql: Relational data processing in spark. In *SIGMOD*, pages 1383–1394, 2015.

[23] ASAP Smoothing in TimescaleDB Toolkit. https://github.com/timescale/timescaledb-toolkit/blob/cbfd6d058cf41591afe2579e9c71ec47c4b095ff/docs/asap.md.

[24] Winda Astuti, Rini Akmeliawati, Wahju Sediono, and MJE Salami. Hybrid technique using singular value decomposition (SVD) and support vector machine (SVM) approach for earthquake prediction. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(5):1719–1728, 2014.

[25] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *International Conference on Similarity Search and Applications*, pages 34–49. Springer, 2017.

[26] Auto-smooth noisy metrics to reveal trends. https://www.datadoghq.com/blog/auto-smoother-asap/. Accessed August 15, 2020.

[27] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, pages 539–550, 2003.

[28] Arturs Backurs, Moses Charikar, Piotr Indyk, and Paris Siminelakis. Efficient Density Evaluation for Smooth Kernels. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 615–626. IEEE, 2018.

[29] P. Bailis, E. Gan, et al. MacroBase: Prioritizing attention in fast data. In *SIGMOD*, pages 541–556, 2017.

[30] P. Bailis, E. Gan, K. Rong, and S. Suri. Prioritizing attention in fast data: Challenges and opportunities. In *CIDR*, 2017.

[31] Shumeet Baluja and Michele Covell. Audio fingerprinting: Combining computer vision & data stream processing. In *IEEE ICASSP*, volume 2, pages II–213–II–216, 2007.

[32] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[33] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web*, pages 651–660, 2005.

[34] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660. ACM, 2005.

[35] Stephen D Bay, Dennis Kibler, Michael J Pazzani, and Padhraic Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *ACM SIGKDD explorations newsletter*, 2(2):81–85, 2000.

[36] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling Up All Pairs Similarity Search. In *WWW*, pages 131–140, 2007.

[37] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331, 1990.

[38] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. *arXiv preprint arXiv:1802.01396*, 2018.

[39] Andrew F. Bell, Stephen Hernandez, H. Elizabeth Gaunt, Patricia Mothes, Mario Ruiz, Daniel Sierra, and Santiago Aguaiza. The rise and fall of periodic 'drumbeat' seismicity at Tungurahua volcano, Ecuador. *Earth and Planetary Science Letters*, 475:58 – 70, 2017.

[40] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[41] Karianne Bergen, Clara Yoon, and Gregory C. Beroza. Scalable Similarity Search in Seismology: A New Approach to Large-Scale Earthquake Detection. *Similarity Search and Applications*, pages 301–308, 2016.

[42] Karianne J Bergen and Gregory C Beroza. Detecting earthquakes over a seismic network using single-station similarity measures. *Geophysical Journal International*, 213(3):1984–1998, 2018.

[43] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[44] Betsy Beyer, Chris Jones, et al., editors. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly, 2016.

[45] Kevin Beyer, Peter J Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*, pages 199–210, 2007.

[46] J A Blackard and D J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, vol.24:131–151, 1999.

[47] Block Sampling in Hive. `https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling`. Accessed: 2020-2-12.

[48] Dmitry Bobrov, Ivan Kitov, and Lassina Zerbo. Perspectives of cross-correlation in seismic monitoring at the international data centre. *Pure and Applied Geophysics*, 171(3):439–468, Mar 2014.

[49] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. Spatio-textual Similarity Joins. *PVLDB*, 6(1):1–12, 2012.

[50] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pages 21–, 1997.

[51] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[52] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 327–336, New York, NY, USA, 1998. Association for Computing Machinery.

[53] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer networks and ISDN systems*, 29(8-13):1157–1166, 1997.

[54] Paul G Brown and Peter J Haas. Techniques for warehousing of sample data. In *ICDE*, pages 6–6, 2006.

[55] Jake Brutlag. Speed matters for google web search, 2009.

[56] Mihai Budiu, Parikshit Gopalan, Lalith Suresh, Udi Wieder, Han Kruiger, and Marcos K Aguilera. Hillview: A trillion-cell spreadsheet for big data. *arXiv preprint arXiv:1907.04827*, 2019.

[57] Stuart K Card, Jock D Mackinlay, and Ben Shneiderman. Information visualization. *Readings in information visualization: using vision to think*, pages 1–34, 1999.

[58] R Chaiken, B Jenkins, P Larson, B Ramsey, D Shakib, S Weaver, and J Zhou. Scope: Easy and e cient parallel processing of massive datasets. *PVLDB*, 1(2):1265–1276, 2008.

[59] FK-P Chan, AW-C Fu, and Clement Yu. Haar wavelets for efficient similarity search of time-series: with and without time warping. *IEEE Transactions on knowledge and data engineering*, 15(3):686–705, 2003.

[60] Badrish Chandramouli, Jonathan Goldstein, and Abdul Quamar. Scalable progressive analytics on big data in the cloud. *PVLDB*, 6(14):1726–1737, 2013.

[61] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[62] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[63] Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1032–1043. IEEE, 2017.

[64] Moses Charikar and Paris Siminelakis. Multi-Resolution Hashing for Fast Pairwise Summations. *arXiv preprint arXiv:1807.07635*, 2018.

[65] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[66] Chris Chatfield. *The analysis of time series: an introduction*. Chapman and Hall/CRC, 2003.

[67] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek Narasayya. Overcoming limitations of sampling for aggregation queries. In *Proceedings 17th International Conference on Data Engineering*, pages 534–542. IEEE, 2001.

[68] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. *SIGMOD Rec.*, 30(2):295–306, 2001.

[69] Surajit Chaudhuri, Gautam Das, and Utkarsh Srivastava. Effective use of block-level sampling in statistics estimation. In *SIGMOD*, pages 287–298, 2004.

[70] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 511–519, 2017.

[71] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. Approximate query processing: No silver bullet. In *SIGMOD*, pages 511–519. ACM, 2017.

[72] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction: How much is enough? *ACM SIGMOD Record*, 27(2):436–447, 1998.

[73] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. *ACM SIGMOD Record*, 28(2):263–274, 1999.

[74] Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *arXiv preprint arXiv:1903.03129*, 2019.

[75] Beidi Chen, Yingchen Xu, and Anshumali Shrivastava. Lsh-sampling breaks the computational chicken-and-egg loop in adaptive stochastic gradient estimation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

[76] Jingdong Chen, J. Benesty, et al. New insights into the noise reduction wiener filter. *TASLP*, pages 1218–1234, 2006.

[77] Yutian Chen, Max Welling, and Alex Smola. Super-samples from Kernel Herding. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, pages 109–116, Arlington, Virginia, United States, 2010. AUAI Press.

[78] Yu Cheng, Weijie Zhao, and Florin Rusu. Bi-level online aggregation on raw data. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2017.

[79] Xiang Ci and Xiaofeng Meng. An efficient block sampling strategy for online aggregation in the cloud. In *International Conference on Web-Age Information Management*, pages 362–373. Springer, 2015.

[80] William S Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, 79(387):531–554, 1984.

[81] Amazon CloudWatch. `https://aws.amazon.com/cloudwatch/`.

[82] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.

[83] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011.

[84] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[85] Graham Cormode and Ke Yi. *Small Summaries for Big Data*. Cambridge University Press, 2020.

[86] Efren Cruz Cortes and Clayton Scott. Sparse Approximation of a Kernel Mean. *Trans. Sig. Proc.*, 65(5):1310–1323, March 2017.

[87] Gualberto Cortés et al. Using Principal Component Analysis to Improve Earthquake Magnitude Prediction in Japan. *Logic Journal of the IGPL*, jzx049:1–14, 10 2017.

[88] Scott Cost and Steven Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine learning*, 10(1):57–78, 1993.

[89] Noel Cressie. *Statistics for spatial data*. John Wiley & Sons, 2015.

[90] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280, 2007.

[91] Gautam Das, Surajit Chaudhuri, and Utkarsh Srivastava. Block-level sampling in statistics estimation, October 6 2005. US Patent App. 10/814,382.

[92] Datadog. `https://www.datadoghq.com/`.

[93] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[94] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 1990.

[95] MCF. de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: A survey. *TVCG*, pages 378–394, 2003.

[96] Lawrence T DeCarlo. On the meaning and use of kurtosis. *Psychological methods*, 2(3):292, 1997.

[97] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[98] Amol Deshpande, Carlos Guestrin, Samuel R Madden, Joseph M Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. *PVLDB*, 30:588–599, 2004.

[99] Designing tables in azure sql data warehouse. https://bit.ly/2T8MsFj.

[100] Luc Devroye and Gary L Wise. Detection of abnormal behavior via nonparametric estimation of the support. *SIAM Journal on Applied Mathematics*, 38(3):480–488, 1980.

[101] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. Sample+ seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*, pages 679–694. ACM, 2016.

[102] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. *PVLDB*, 1(2):1542–1552, 2008.

[103] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586, 2011.

[104] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling lsh for performance tuning. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 669–678, 2008.

[105] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling LSH for Performance Tuning. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 669–678, 2008.

[106] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 1973.

[107] Clara E. Yoon, Yihe Huang, William L. Ellsworth, and Gregory C. Beroza. Seismicity During the Initial Stages of the Guy-Greenbrier, Arkansas, Earthquake Sequence. *Journal of Geophysical Research: Solid Earth*, 122(11):9253–9274, 2017.

[108] Clara E Yoon, Ossian O'Reilly, Karianne Bergen, and Gregory C Beroza. Earthquake detection through computationally efficient similarity search. *Science Advances*, 1(11):e1501057, 2015.

[109] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[110] E Robert Engdahl, Rob van der Hilst, and Raymond Buland. Global teleseismic earthquake relocation with improved travel times and procedures for depth determination. *Bulletin of the Seismological Society of America*, 88(3):722–743, 1998.

[111] Arnab Nandi Eugene Wu. Towards perception-aware interactive data visualization systems. In *DSIA*, 2015.

[112] FALCONN - FAst Lookups of Cosine and Other Nearest Neighbors. `https://github.com/falconn-lib/falconn`.

[113] Christos Faloutsos, Ron Barber, Myron Flickner, Jim Hafner, Wayne Niblack, Dragutin Petkovic, and William Equitz. Efficient and effective querying by image content. *Journal of intelligent information systems*, 3(3-4):231–262, 1994.

[114] FAST Detection Pipeline. `https://github.com/stanford-futuredata/FAST`.

[115] Raphael A Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[116] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[117] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

[118] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *PVLDB*, 12(5):461–474, 2019.

[119] Tak-chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[120] Tak-chung Fu, Fu-lai Chung, Robert Luk, and Chak-man Ng. Representing financial time series based on data point importance. *Engineering Applications of Artificial Intelligence*, pages 277 – 300, 2008.

[121] Wayne A. Fuller. *Probability Sampling from a Finite Universe*, chapter 1, pages 1–93. John Wiley & Sons, Ltd, 2009.

[122] Mélissa Gaillard and Stefania Pandolfi. Cern data centre passes the 200-petabyte milestone.(jul 2017). *URL http://cds. cern. ch/record/2276551*, 2017.

[123] Edward Gan and Peter Bailis. Scalable kernel density classification via threshold-based pruning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 945–959. ACM, 2017.

[124] Edward Gan, Peter Bailis, and Moses Charikar. Coopstore: Optimizing precomputed summaries for aggregation. *PVLDB*, 13(11):2174–2187, 2020.

[125] Ganglia Monitoring System. http://ganglia.info/.

[126] John S Garofolo. TIMIT acoustic phonetic continuous speech corpus. *Linguistic Data Consortium, 1993*, 1993.

[127] Robert J. Geller and Charles S. Mueller. Four similar earthquakes in central california. *Geophysical Research Letters*, 7(10):821–824, 1980.

[128] Rainer Gemulla, Wolfgang Lehner, and Peter J Haas. Maintaining bernoulli samples over evolving multisets. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 93–102, 2007.

[129] GeoNet. https://www.geonet.org.nz/data/tools/FDSN.

[130] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

[131] Phillip B Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 331–342, 1998.

[132] Steven J. Gibbons and Frode Ringdal. The detection of low magnitude seismic events using array-based waveform correlation. *Geophysical Journal International*, 165(1):149–166, 2006.

[133] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *PVLDB*, page 518–529, 1999.

[134] Lewis Girod, Kyle Jamieson, et al. Wavescope: a signal-oriented data stream management system. In *SenSys*, pages 421–422, 2006.

[135] Graphite. https://graphiteapp.org/.

[136] Alexander G Gray and Andrew W Moore. N-body'problems in statistical learning. In *Advances in neural information processing systems*, pages 521–527, 2001.

[137] Alexander G Gray and Andrew W Moore. Nonparametric density estimation: Toward computational tractability. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 203–211. SIAM, 2003.

[138] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[139] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.

[140] Yu Jeffrey Gu, Ahmet Okeler, Sean Contenti, Kenny Kocon, Luyi Shen, and Keith Brzak. Broadband seismic array deployment and data analysis in Alberta. *CSEG Recorder, September*, pages 37–44, 2009.

[141] B. GUTENBERG and C. F. RICHTER. Magnitude and energy of earthquakes. *Annals of Geophysics*, 9(1):1–15, 1956.

[142] Peter J Haas and Joseph M Hellerstein. Ripple joins for online aggregation. *ACM SIGMOD Record*, 28(2):287–298, 1999.

[143] Peter J. Haas and Christian König. A bi-level bernoulli scheme for database sampling. In *SIGMOD*, pages 275–286, 2004.

[144] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[145] Alon Y Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[146] Alon Y Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[147] John M Hammersley and DC Handscomb. Percolation processes. In *Monte Carlo Methods*, pages 134–141. Springer, 1964.

[148] Sangjin Han, Norbert Egi, Aurojit Panda, Sylvia Ratnasamy, Guangyu Shi, and Scott Shenker. Network support for resource disaggregation in next-generation datacenters. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pages 1–7, 2013.

[149] Venky Harinarayan, Anand Rajaraman, and Jeffrey D Ullman. Implementing data cubes efficiently. *Acm Sigmod Record*, 25(2):205–216, 1996.

[150] Ben Harwood and Tom Drummond. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5713–5722, 2016.

[151] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. Online aggregation. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 171–182, 1997.

[152] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. Online aggregation. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 171–182, 1997.

[153] Amina Helmi and P Tim de Zeeuw. Mapping the substructure in the galactic halo with the next generation of astrometric satellites. *Monthly Notices of the Royal Astronomical Society*, 319(3):657–665, 2000.

[154] Robert Hendron and Cheryn Engebrecht. Building america research benchmark definition: Updated december 2009, 2010.

[155] Alexander Hinneburg and Hans-Henning Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In Michael R. Berthold, John Shawe-Taylor, and Nada Lavrač, editors, *Advances in Intelligent Data Analysis VII*, pages 70–80, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[156] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, pages 1–18, 2004.

[157] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[158] Jake Hofman, Daniel G. Goldstein, and Jessica Hullman. How visualizing inferential uncertainty can mislead readers about treatment effects in scientific results. In *CHI 2020*. ACM, April 2020.

[159] Wen-Chi Hou and Gultekin Ozsoyoglu. Statistical estimators for aggregate relational algebra queries. *ACM Transactions on Database Systems (TODS)*, 16(4):600–654, 1991.

[160] Michael Httermann. *DevOps for developers*. Apress, 2012.

[161] Yihe Huang and Gregory C. Beroza. Temporal variation in the magnitude-frequency distribution during the Guy-Greenbrier earthquake sequence. *Geophysical Research Letters*, 42(16):6639–6646, 2015.

[162] Mans Hulden, Miikka Silfverberg, and Jerid Francom. Kernel density estimation for text-based geolocation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 145–150. AAAI Press, 2015.

[163] Vedad Hulusić, Gabriela Czanner, et al. Investigation of the beat rate effect on frame rate for animated content. In *SCCG*, pages 151–159, 2009.

[164] R.J. Hyndman. Time series data library. http://data.is/TSDLdemo.

[165] Stratos Idreos, Martin L Kersten, Stefan Manegold, et al. Database cracking. In *CIDR*, volume 7, pages 68–78, 2007.

[166] Stratos Idreos, Stefan Manegold, Harumi Kuno, and Goetz Graefe. Merging what's cracked, cracking what's merged: adaptive indexing in main-memory column-stores. *PVLDB*, 4(9):586–597, 2011.

[167] Impala Partition Pruning. `https://docs.cloudera.com/runtime/7.2.10/impala-reference/topics/impala-partition-pruning.html`. Accessed: 2021-8-17.

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

[169] Christopher Jermaine, Abhijit Pol, and Subramanian Arumugam. Online maintenance of very large random samples. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 299–310, 2004.

[170] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang. Min-Max Hash for Jaccard Similarity. In *2013 IEEE 13th International Conference on Data Mining*, pages 301–309, 2013.

[171] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *CoRR*, abs/1702.08734, 2017.

[172] Manfred Joswig. Pattern recognition for earthquake detection. *Bulletin of the Seismological Society of America*, 80(1):170, 1990.

[173] Uwe Jugel, Zbigniew Jerzak, and other. M4: A visualization-oriented time series data aggregation. In *VLDB*, pages 797–808, 2014.

[174] K. Rong, Y. Lu, P. Bailis, S. Kandula, P. Levis. Approximate Partition Selection for Big-Data Workloads using Summary Statistics (Extended Version). `https://kexinrong.github.io/papers/ps3.pdf`. Accessed: 2020-8-20.

[175] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey 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.

[176] Srikanth Kandula, Kukjin Lee, Surajit Chaudhuri, and Marc Friedman. Experiences with approximating queries in microsoft's production big-data clusters. *PVLDB*, 12(12):2131–2142, 2019.

[177] Srikanth Kandula, Kukjin Lee, Surajit Chaudhuri, and Marc Friedman. Experiences with approximating queries in microsoft's production big-data clusters. *PVLDB*, 12(12):2131–2142, 2019.

[178] Srikanth Kandula, Laurel Orr, and Surajit Chaudhuri. Pushing data-induced predicates through joins in big-data clusters. *PVLDB*, 13(3):252–265, 2019.

[179] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*, pages 631–646, 2016.

[180] Byungkon Kang and Kyomin Jung. Robust and efficient locality sensitive hashing for nearest neighbor search in large data sets. In *NIPS Workshop on Big Learning (BigLearn)*, pages 1–8, 2012.

[181] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*, pages 71–78. IEEE, 2016.

[182] Aitaro Kato and Shigeki Nakagawa. Multiple slow-slip events during a foreshock sequence of the 2014 Iquique, Chile Mw 8.1 earthquake. *Geophysical Research Letters*, 41(15):5420–5427, 2014.

[183] Yannis Katsis, Yoav Freund, and Yannis Papakonstantinou. Combining databases and signal processing in plato. In *CIDR*, 2015.

[184] Eamonn Keogh et al. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, pages 263–286, 2001.

[185] Eamonn Keogh et al. Finding surprising patterns in a time series database in linear time and space. In *KDD*, pages 550–556, 2002.

[186] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, 7(4):349–371, 2003.

[187] Eamonn Keogh, Jessica Lin, and Ada Fu. HOT SAX: Efficiently finding the most unusual time series subsequence. In *ICDM*, pages 226–233, 2005.

[188] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[189] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[190] Qingkai Kong, Richard M. Allen, Louis Schreier, and Young-Woo Kwon. MyShake: A smartphone seismic network for earthquake early warning and beyond. *Science Advances*, 2(2):e1501055, 2016.

[191] Tim Kraska. Northstar: An interactive data science system. *PVLDB*, 11(12):2150–2164, 2018.

[192] Erwin Kreyszig. *Advanced Engineering Mathematics*. Wiley, NY, fourth edition, 1979.

[193] Raja Kulkarni. A Review Of Application Of Data Mining In Earthquake Prediction. In *International Journal of Computer Science and Information Technology (IJCSIT)*, 2012.

[194] Thijs Laarhoven. Graph-Based Time-Space Trade-Offs for Approximate Near Neighbors. In *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:14, 2018.

[195] Ove Daae Lampe and Helwig Hauser. Interactive visualization of streaming data with kernel density estimation. In *2011 IEEE Pacific visualization symposium*, pages 171–178. IEEE, 2011.

[196] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. Outlier detection with kernel density functions. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 61–75. Springer, 2007.

[197] A. Lavin and S. Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In *IEEE ICMLA*, pages 38–44, 2015.

[198] Dongryeol Lee and Alexander G Gray. Fast high-dimensional kernel summations using the monte carlo multipole method. In *Advances in Neural Information Processing Systems*, pages 929–936, 2009.

[199] Dongryeol Lee, Andrew W Moore, and Alexander G Gray. Dual-tree fast gauss transforms. In *Advances in Neural Information Processing Systems*, pages 747–754, 2006.

[200] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[201] Jin Li et al. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, pages 39–44, 2005.

[202] Kaiyu Li and Guoliang Li. Approximate query processing: What is new and where to go? *Data Science and Engineering*, 3(4):379–397, 2018.

[203] Lisha Li, Kevin Jamieson, et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv:1603.06560*, 2016.

[204] T Warren Liao. Clustering of time series data–a survey. *Pattern recognition*, 38(11):1857–1874, 2005.

[205] T. Warren Liao. Clustering of time series data: a survey. *Pattern Recognition*, pages 1857–1874, 2005.

[206] M. Lichman. UCI machine learning repository, 2013. Accessed 19-Aug-2016.

[207] Jessica Lin, Eamonn Keogh, et al. Visually mining and monitoring massive time series. In *KDD*, pages 460–469, 2004.

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

[209] Chen Luo and Anshumali Shrivastava. Arrays of (locality-sensitive) Count Estimators (ACE): Anomaly Detection on the Edge. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1439–1448. International World Wide Web Conferences Steering Committee, 2018.

[210] Chen Luo and Anshumali Shrivastava. Scaling-up Split-Merge MCMC with Locality Sensitive Sampling (LSS). *arXiv preprint arXiv:1802.07444*, 2018.

[211] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: Efficient Indexing for High-dimensional Similarity Search. *VLDB*, pages 950–961, 2007.

[212] Jock Mackinlay. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics (Tog)*, 5(2):110–141, 1986.

[213] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. Show me: Automatic presentation for visual analysis. *TVCG*, pages 1137–1144, 2007.

[214] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[215] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

[216] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150, 2007.

[217] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *PVLDB*, pages 346–357, 2002.

[218] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. An Empirical Evaluation of Set Similarity Join Techniques. *PVLDB*, 9(9):636–647, 2016.

[219] W. March, B. Xiao, and G. Biros. ASKIT: Approximate Skeletonization Kernel-Independent Treecode in High Dimensions. *SIAM Journal on Scientific Computing*, 37(2):A1089–A1110, 2015.

[220] W. March, B. Xiao, C. Yu, and G. Biros. ASKIT: An Efficient, Parallel Library for High-Dimensional Kernel Summations. *SIAM Journal on Scientific Computing*, 38(5):S720–S749, 2016.

[221] J. S. Marron. Automatic smoothing parameter selection: A survey. *Empirical Economics*, 13(3):187–208, 1988.

[222] John C McCallum. Disk Drive Prices 1955+. https://jcmit.net/diskprice.htm. Accessed July 23, 2021.

[223] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems (TODS)*, 31(3):1095–1133, 2006.

[224] Microsoft Azure Monitor. https://docs.microsoft.com/azure/monitoring-and-diagnostics.

[225] Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[226] Vlad I. Morariu, Balaji V. Srinivasan, Vikas C Raykar, Ramani Duraiswami, and Larry S Davis. Automatic online tuning for fast Gaussian summation. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1113–1120. Curran Associates, Inc., 2009.

[227] Kristi Morton, Magdalena Balazinska, Dan Grossman, and Jock Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *PVLDB*, 7(6):453–456, 2014.

[228] Yoichi Murakami and Kenji Mizuguchi. Applying the Naïve Bayes classifier with kernel density estimation to the prediction of protein–protein interaction sites. *Bioinformatics*, 26(15):1841–1848, 06 2010.

[229] Shanmugavelayutham Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[230] MySQL Partition Pruning. https://dev.mysql.com/doc/mysql-partitioning-excerpt/8.0/en/partitioning-pruning.html. Accessed: 2020-2-12.

[231] Ján Jakub Naništa. Downsampling methods for time series visualisation. https://www.npmjs.com/package/downsample. Accessed October 25, 2020.

[232] Suman Nath and Phillip B. Gibbons. Online maintenance of very large random samples on flash storage. In *PVLDB*, pages 970–983, 2008.

[233] NCEDC. http://service.ncedc.org/.

[234] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS workshop on deep learning and unsupervised feature learning*, 2011.

[235] New Relic. https://newrelic.com/.

[236] M.S. Nikulin. Excess coefficient. In Michiel Hazewinkel, editor, *Encyclopedia of Mathematics*. Kluwer Academic Publishers, 2002.

[237] Frank Olken. *Random sampling from databases*. PhD thesis, University of California, Berkeley, 1993.

[238] Data Warehousing Guide: Using Zone Maps. https://docs.oracle.com/en/database/oracle/oracle-database/21/dwhsg/using-zone-maps.html. Accessed: 2021-8-17.

[239] Oracle Database optimizer statistics. https://docs.oracle.com/en/database/oracle/oracle-database/18/tgsql/optimizer-statistics-concepts.htm. Accessed: 2020-2-12.

[240] Niketan Pansare, Vinayak Borkar, Chris Jermaine, and Tyson Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4(11):1135–1145, 2011.

[241] Niketan Pansare, Vinayak R Borkar, Chris Jermaine, and Tyson Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4(11):1135–1145, 2011.

[242] Yongjoo Park, Ahmad Shahab Tajik, Michael Cafarella, and Barzan Mozafari. Database learning: Toward a database that becomes smarter every time. In *SIGMOD*, pages 587–602, 2017.

[243] Tuomas Pelkonen et al. Gorilla: A fast, scalable, in-memory time series database. In *VLDB*, pages 1816–1827, 2015.

[244] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *PVLDB*, 8(12):1816–1827, 2015.

[245] Jinglin Peng, Hongzhi Wang, Jianzhong Li, and Hong Gao. Set-based similarity search for time series. In *SIGMOD*, pages 2039–2052, 2016.

[246] Jinglin Peng, Dongxiang Zhang, Jiannan Wang, and Jian Pei. Aqp++ connecting approximate query processing with aggregate precomputation for interactive analytics. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1477–1492, 2018.

[247] Jinglin Peng, Dongxiang Zhang, Jiannan Wang, and Jian Pei. Aqp++: connecting approximate query processing with aggregate precomputation for interactive analytics. In *SIGMOD*, pages 1477–1492. ACM, 2018.

[248] Zhigang Peng and Peng Zhao. Migration of early aftershocks following the 2004 Parkfield earthquake. *Nature Geoscience*, 2:877 EP –, 2009.

[249] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[250] Thibaut Perol, Michaël Gharbi, and Marine Denolle. Convolutional neural network for earthquake detection and location. *Science Advances*, 4(2), 2018.

[251] Jeff M Phillips. ε-samples for kernels. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1622–1632. SIAM, 2013.

[252] Jeff M Phillips and Wai Ming Tai. Near-optimal coresets of kernel density estimates. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 99. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[253] PostgreSQL 9.5.21 TABLESAMPLE. https://www.postgresql.org/docs/9.5/sql-select.html. Accessed: 2020-2-12.

[254] William H. Press, Saul A. Teukolsky, et al. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. Cambridge University Press, 1992.

[255] Processing Petabytes of Data in Seconds with Databricks Delta. https://databricks.com/blog/2018/07/31/processing-petabytes-of-data-in-seconds-with-databricks-delta.html. Accessed: 2021-8-17.

[256] Program for TPC-H Data Generation with Skew. https://www.microsoft.com/en-us/download/details.aspx?id=52430. Accessed: 2020-2-12.

[257] Liudmila Prokhorenkova and Aleksandr Shekhovtsov. Graph-based nearest neighbor search: From practice to theory. In *International Conference on Machine Learning*, pages 7803–7813. PMLR, 2020.

[258] Prometheus. https://prometheus.io/.

[259] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[260] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*, pages 262–270, 2012.

[261] Parikshit Ram, Dongryeol Lee, William March, and Alexander G Gray. Linear-time algorithms for pairwise statistical problems. In *Advances in Neural Information Processing Systems*, pages 1527–1535, 2009.

[262] BiChen Rao and Erkang Zhu. Searching Web Data Using MinHash LSH. In *SIGMOD*, pages 2257–2258, 2016.

[263] David Reinsel, John Rydning, and John F. Gantz. Worldwide Global DataSphere Forecast, 2021–2025: The World Keeps Creating More Data - Now, What Do We Do with It All? `https://www.idc.com/getdoc.jsp?containerId=US46410421`, March 2021.

[264] H. L. Resnikoff. *The illusion of reality / Howard L. Resnikoff.* Springer-Verlag, New York, 1989 - 1989.

[265] K. Reumann and A. P. M. Witkam. Optimizing curve segmentation in computer graphics. *ICS*, 1974.

[266] Kexin Rong and Peter Bailis. Asap: Prioritizing attention via time series smoothing. *PVLDB*, 10(11):1358–1369, August 2017.

[267] Kexin Rong and Peter Bailis. ASAP: Prioritizing attention via time series smoothing (extended version). *arXiv:1703.00983*, 2017.

[268] Kexin Rong, Yao Lu, Peter Bailis, Srikanth Kandula, and Philip Levis. Approximate partition selection for big-data workloads using summary statistics. *PVLDB*, 13(11):2606–2619, 2020.

[269] Kexin Rong, Clara E Yoon, Karianne J Bergen, Hashem Elezabi, Peter Bailis, Philip Levis, and Gregory C Beroza. Locality-sensitive hashing for earthquake detection: A case study of scaling data-driven science. *PVLDB*, 11(11):1674–1687, 2018.

[270] Kexin Rong, Clara E. Yoon, Karianne J. Bergen, Hashem Elezabi, Peter Bailis, Philip Levis, and Gregory C. Beroza. Locality-sensitive hashing for earthquake detection: A case study of scaling data-driven science (extended version). *arXiv:1803.09835*, 2018.

[271] JE Rossiter. Calculating centile curves using kernel density estimation methods with application to infant kidney lengths. *Statistics in Medicine*, 10(11):1693–1701, 1991.

[272] Karl Rupp. 48 years of microprocessor trend data, July 2020. 2020 update of the popular chart hosted at https://github.com/karlrupp/microprocessor-trend- data.

[273] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes Twitter users: real-time event detection by social sensors. In *WWW*, pages 851–860, 2010.

[274] Stan Salvador and Philip Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. *Tools with Artificial Intelligence*, pages 576–584, 2004.

[275] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[276] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 1964.

[277] Patrick Schäfer and Ulf Leser. Fast and Accurate Time Series Classification with WEASEL. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 637–646, 2017.

[278] David P. Schaff and Gregory C. Beroza. Coseismic and postseismic velocity changes measured by repeating earthquakes. *Journal of Geophysical Research: Solid Earth*, 109(B10), 2004.

[279] David P. Schaff and Felix Waldhauser. One Magnitude Unit Reduction in Detection Threshold by Cross Correlation Applied to Parkfield (California) and China SeismicityOne Magnitude Unit Reduction in Detection Threshold by Cross Correlation. *Bulletin of the Seismological Society of America*, 100(6):3224, 2010.

[280] Felix Martin Schuhknecht, Alekh Jindal, and Jens Dittrich. The uncracked pieces in database cracking. *PVLDB*, 7(2):97–108, 2013.

[281] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[282] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *PVLDB*, page 507–518, 1987.

[283] S Seshadri and Jeffrey F Naughton. Sampling issues in parallel database systems. In *International Conference on Extending Database Technology*, pages 328–343. Springer, 1992.

[284] David R. Shelly, David P. Hill, Frédérick Massin, Jamie Farrell, Robert B. Smith, and Taka'aki Taira. A fluid-driven earthquake swarm on the margin of the Yellowstone caldera. *Journal of Geophysical Research: Solid Earth*, 118(9):4872–4886, 2013.

[285] W. Shi and B. M. Golam Kibria. On some confidence intervals for estimating the mean of a skewed population. *International Journal of Mathematical Education in Science and Technology*, 38(3):412–421, 2007.

[286] Wenzhong Shi and ChuiKwan Cheung. Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, pages 27–44, 2006.

[287] Yingjie Shi, Xiaofeng Meng, Fusheng Wang, and Yantao Gan. You can stop early with cola: online processing of aggregate queries in the cloud. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1223–1232. ACM, 2012.

[288] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *arXiv preprint arXiv:1405.5869*, 2014.

[289] Anshumali Shrivastava and Ping Li. In Defense of MinHash Over SimHash. In *AISTATS*, volume 33, Reykjavik, Iceland, 2014.

[290] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.

[291] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications*. Springer, 2005.

[292] Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. F1: A distributed sql database that scales. In *VLDB*, 2013.

[293] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010.

[294] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. Effortless visual data exploration with zenvisage: An interactive and expressive visual analytics system. In *VLDB*, pages 457 – 468, 2017.

[295] Paris Siminelakis, Kexin Rong, Peter Bailis, Moses Charikar, and Philip Levis. Rehashing kernel evaluation in high dimensions. In *International Conference on Machine Learning*, pages 5789–5798. PMLR, 2019.

[296] Herbert A Simon. Designing organizations for an information-rich world. *International Library of Critical Writings in Economics*, 70:187–202, 1996.

[297] Julius O. Smith. *Spectral Audio Signal Processing*. W3K, 2011.

[298] Steven W. Smith. CHAPTER 15 - moving average filters. In *Digital Signal Processing*. Elsevier, 2003.

[299] Snowflake SAMPLE / TABLESAMPLE. `https://docs.snowflake.net/manuals/sql-reference/constructs/sample.html`. Accessed: 2020-2-12.

[300] Ryan Spring and Anshumali Shrivastava. A new unbiased and efficient class of lsh-based samplers and estimators for partition function computation in log-linear models. *arXiv preprint arXiv:1703.05160*, 2017.

[301] Ryan Spring and Anshumali Shrivastava. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 445–454, 2017.

[302] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Jose M Faleiro, Joseph E Gonzalez, Joseph M Hellerstein, and Alexey Tumanov. Cloudburst: Stateful functions-as-a-service. *arXiv preprint arXiv:2001.04592*, 2020.

[303] Bharath Sriperumbudur et al. On the optimal estimation of probability measures in weak and strong topologies. *Bernoulli*, 22(3):1839–1893, 2016.

[304] Google Stackdriver. `https://cloud.google.com/stackdriver/`.

[305] Liwen Sun, Michael J Franklin, Sanjay Krishnan, and Reynold S Xin. Fine-grained partitioning for aggressive data skipping. In *SIGMOD*, pages 1115–1126, 2014.

[306] Liwen Sun, Michael J Franklin, Sanjay Krishnan, and Reynold S Xin. Fine-grained partitioning for aggressive data skipping. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1115–1126, 2014.

[307] Liwen Sun, Michael J Franklin, Jiannan Wang, and Eugene Wu. Skipping-oriented partitioning for columnar layouts. *PVLDB*, 10(4):421–432, 2016.

[308] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming Similarity Search over One Billion Tweets Using Parallel Locality-sensitive Hashing. *PVLDB*, 6(14):1930–1941, 2013.

[309] Fumio Tajima. Determination of window size for analyzing dna sequences. *Journal of Molecular Evolution*, pages 470–473, 1991.

[310] Kanat Tangwongsan et al. General incremental sliding-window aggregation. In *VLDB*, pages 702–713, 2015.

[311] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV*, pages 839–846, 1998.

[312] Thanh N. Tran, Ron Wehrens, and Lutgarde M.C. Buydens. Knn-kernel density-based clustering for high-dimensional multivariate data. *Computational Statistics and Data Analysis*, 51(2):513–525, 2006.

[313] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, USA, 1986.

[314] John W Tukey et al. *Exploratory data analysis*, volume 2. Reading, Mass., 1977.

[315] Oliver Thorsten Unke and Markus Meuwly. Kernel density estimation-based solution of the nuclear schrödinger equation. *Chemical Physics Letters*, 639:52–56, 2015.

[316] Manasi Vartak, Silu Huang, Tarique Siddiqui, Samuel Madden, and Aditya Parameswaran. Towards visualization recommendation systems. *ACM SIGMOD Record*, 45(4):34–39, 2017.

[317] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[318] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, pages 46–51, 1993.

[319] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[320] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. Building an elastic query engine on disaggregated storage. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 449–462, 2020.

[321] Brett Walenz, Stavros Sintos, Sudeepa Roy, and Jun Yang. Learning to sample: Counting with complex queries. *PVLDB*, 13(3):390–402, 2019.

[322] Jiannan Wang, Guoliang Li, and Jianhua Feng. Can We Beat the Prefix Filtering? An Adaptive Framework for Similarity Join and Search. In *SIGMOD*, pages 85–96, 2012.

[323] Jin Wang and Ta-liang Teng. Identification and picking of S phase using an artificial neural network. *Bulletin of the Seismological Society of America*, 87(5):1140, 1997.

[324] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

[325] Spencer Weart. The carbon dioxide greenhouse effect. *The Discovery of Global Warming*, 2008.

[326] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, volume 98, pages 194–205, 1998.

[327] Peter H Westfall. Kurtosis as Peakedness, 1905–2014. RIP. *The American Statistician*, pages 191–195, 2014.

[328] Mitchell Withers, Richard Aster, Christopher Young, Judy Beiriger, Mark Harris, Susan Moore, and Julian Trujillo. A comparison of select trigger algorithms for automated global seismic phase and event detection. *Bulletin of the Seismological Society of America*, 88(1):95, 1998.

[329] Kanit Wongsuphasawat et al. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *TVCG*, pages 649–658, 2016.

[330] Eugene Wu, Leilani Battle, and Samuel R Madden. The case for data visualization management systems: vision paper. In *VLDB*, pages 903–906, 2014.

[331] XGBoost Feature Importance. `https://xgboost.readthedocs.io/en/latest/python/python_api.html`. Accessed: 2020-2-12.

[332] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient Similarity Joins for Near-duplicate Detection. *TODS*, 36(3):15:1–15:41, 2011.

[333] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A Brief Survey on Sequence Classification. *SIGKDD Explor. Newsl.*, 12(1):40–48, November 2010.

[334] Zongheng Yang, Badrish Chandramouli, Chi Wang, Johannes Gehrke, Yinan Li, Umar Farooq Minhas, Per-Åke Larson, Donald Kossmann, and Rajeev Acharya. Qd-tree: Learning data layouts for big data analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 193–208, 2020.

[335] Zongheng Yang, Badrish Chandramouli, Chi Wang, Johannes Gehrke, Yinan Li, Umar Farooq Minhas, Per-Åke Larson, Donald Kossmann, and Rajeev Acharya. Qd-tree: Learning data layouts for big data analytics. In *SIGMOD*, page 193–208, 2020.

[336] Efstratios Ydraios et al. *Database cracking: towards auto-tunning database kernels*. SIKS, 2010.

[337] Byoung-Kee Yi and Christos Faloutsos. Fast Time Sequence Indexing for Arbitrary Lp Norms. *VLDB*, pages 385–394, 2000.

[338] Clara Yoon, Karianne Bergen, Kexin Rong, Hashem Elezabi, William L Ellsworth, Gregory C Beroza, Peter Bailis, and Philip Levis. Unsupervised large-scale search for similar earthquake signals. *Bulletin of the Seismological Society of America*, 109(4):1451–1468, 2019.

[339] Erfan Zamanian, Carsten Binnig, and Abdallah Salama. Locality-aware partitioning in parallel database systems. In *SIGMOD*, pages 17–30, 2015.

[340] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1525–1539, 2018.

[341] Denys Zhdanov. What's new in graphite 1.1. `https://archive.fosdem.org/2018/schedule/event/whats_new_in_graphite_11/`. FOSDEM 2018. Accessed September 5, 2020.

[342] Yan Zheng, Jeffrey Jestes, Jeff M Phillips, and Feifei Li. Quality and efficiency for kernel density estimates in large data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 433–444. ACM, 2013.

[343] Jingren Zhou, Nicolas Bruno, Ming-Chuan Wu, Per-Ake Larson, Ronnie Chaiken, and Darren Shakib. Scope: parallel databases meet mapreduce. *PVLDB*, 21(5):611–636, 2012.

[344] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *KDD*, pages 336–345, 2003.