Efficient Approximate Query Processing with Block Sampling

Yuxuan Zhu
yxx404@illinois.edu
University of Illinois Urbana-Champaign

Daniel Kang
ddkang@illinois.edu
University of Illinois Urbana-Champaign

ABSTRACT

Approximate query processing (AQP) has been widely studied to accelerate online analytical query processing while maintaining high accuracy. Many existing methods focus on reducing data processing costs through record-level sampling techniques. However, since data systems typically access data in pages, these methods can cause data loading costs as high as exact queries, often becoming the bottleneck of query processing. In this work, we present B-AQP, an AQP framework based on block sampling, significantly reducing data loading costs while guaranteeing a priori errors. Our preliminary evaluation across various data systems and workloads demonstrates that B-AQP accelerates query execution by up to 185× compared to uniform sampling and four orders of magnitude compared to exact queries, all with guaranteed errors.

1 INTRODUCTION

Loading data from storage mediums to memory is often necessary but can be expensive for many analytical queries, especially when dealing with data larger than memory or cold data [5, 15]. Quantitatively, data loading of the TPC-H benchmark accounts for 84% of query processing time in DuckDB.[1] Even for computationally demanding tasks, such as machine learning analytics, data loading can remain the most time-consuming part when data are stored in remote storage with limited bandwidth [15, 22]. This is particularly true when accelerators (e.g., NVIDIA H100) are available. Our evaluation shows that accessing Parquet-based textual datasets from Amazon S3 takes 64% of the query time.[2]

Online approximate query processing (AQP) aims to speed up query processing and reduce computational costs by minimizing the amount of data that needs to be processed. However, current online AQP methods fail to achieve efficient data loading in their sampling algorithms. These algorithms typically operate at the level of individual data records to minimize sample size [13, 14, 20]. In contrast, modern data systems access data in pages to reduce the overhead of I/O operations. Consequently, sampling a single random record can require accessing an entire data page, which is inefficient and time-consuming for online query processing. We show an example in Figure 1, where uniform sampling needs to access most of the data pages to achieve a 5% error.

By better matching the data access patterns of storage systems, directly sampling data pages (i.e., block sampling) can significantly save data loading costs (Figure 1). However, prior algorithms that leverage block sampling do not provide a priori error guarantees, resulting in limited adoptions in the real-world [6, 7, 9, 11, 19].

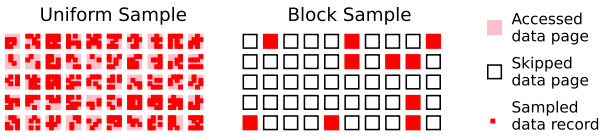


Figure 1: Accessed data pages when using uniform sampling or block sampling to approximately answer an AVG query over a Parquet file. Targeting the same 5% error, block sampling reduces data loading by 5×.

In this work, we propose a novel online AQP framework: B-AQP, which achieves significant cost reductions using block sampling and ensures a priori error bounds for approximate results. B-AQP improves data loading efficiency for both structured and unstructured data, without requiring modifications to existing data systems or extra maintenance.

Accurate error estimation is the primary challenge when integrating block sampling into query processing. Block sampling introduces dependencies among data records within the same block, creating statistical challenges in estimating the error of the approximate query. To illustrate, consider the following ML query that counts the number of sentiment-positive reviews:

```
SELECT COUNT(*) FROM reviews WHERE sentiment = 'positive'
```

If we target a 5% error and directly apply the Central Limit Theorem (CLT) for uniform samples on the block sampling with a block size of 100, we would obtain an average error of 8.5% and a 95th percentile error of 20% over 1000 runs.

To address this challenge and achieve a priori error guarantees, we design a novel statistical engine based on block-oriented estimators and pilot sampling. Specifically, we estimate aggregates of data records using independent and identically distributed block-level statistics. In this way, we can obtain valid confidence intervals via CLT. To achieve a priori error guarantees, we apply pilot sampling to analyze the required sample size for the given error specification, ensuring the final estimation error remains below the target error. Finally, we theoretically analyze that B-AQP can achieve significant cost reductions compared to uniform sampling when the page size is large or data pages are representative of the entire dataset.

We implement B-AQP as middleware for both on-premise and cloud data systems. Our preliminary evaluations across structured and unstructured data show that B-AQP accelerates query execution by up to four orders of magnitude (1.5-12,412×) compared to exact queries and up to 185× compared to uniform sampling, while consistently guaranteeing a priori errors.

[1] We used a scale factor of 5,000, two 32-core AMD 7543 CPUs, and 256 GB RAM.
[2] We uniformly sampled 1% of 1 TB data and executed queries on 8 H100 GPUs, a BCM 5720 Gigabit NIC with a finetuned roBERTa model [4].

 15th Annual Conference on Innovative Data Systems Research (CIDR '25). January 19-22, Amsterdam, The Netherlands

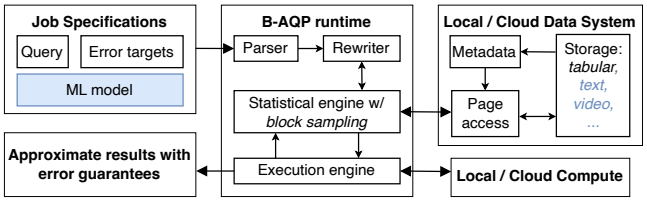


Figure 2: B-AQP architecture. Blue components are for ML analytics over unstructured data.

2 OVERVIEW

2.1 System Architecture

Figure 2 shows the architecture of B-AQP, consisting of a middleware B-AQP runtime and multiple interfaces for users, data systems, and computing resources.

User Interface. B-AQP requires users to specify two inputs: (1) an aggregation query described in SQL and (2) a target error of the result, along with a target probability indicating the probability of meeting the target error. For ML analytics over unstructured data, B-AQP requires users to specify the ML models or APIs used in the query. B-AQP returns approximate results that are guaranteed to satisfy the error specification.

Data Interface. B-AQP requires access to the minimum indivisible storage unit (e.g., a data page). In analytical database management systems, such as DuckDB, this can be done via the `TABLESAMPLE SYSTEM` clause. In cloud data systems, such as Amazon S3, we can read pages by requesting specific byte ranges, given the metadata.

B-AQP runtime. B-AQP executes a runtime that processes incoming jobs and communicates with users and other systems. We describe the basic components that ensure the functionality of B-AQP, while customized block sampling algorithms can be further developed to better fit specific workloads. B-AQP is composed of a *rewriter* that rewrites queries with page-level data interfaces, a *statistical engine* that samples data and analyzes errors, and an *execution engine* that processes the sampled data. We describe the interactions between these components in the next section.

2.2 Algorithm Overview

In B-AQP, we apply block sampling to reduce data loading costs and design a two-stage sampling algorithm to achieve error guarantees. In the first stage, B-AQP rewrites the input query into a pilot query that computes statistics and determines the sample size for the error specification. In the second stage, B-AQP rewrites the input query into a final query that computes the approximate answer. For both stages, B-AQP materializes the sampled blocks using data page access APIs and computes rewritten queries using data process APIs. Algorithm 1 illustrates the procedures.

2.3 Use Cases

We highlight two cases where B-AQP can be applied: online analytical processing [1] and cloud-native ML analytics [2].

Online Analytical Processing. Although computing aggregation over structured data is fast on modern hardware, loading data and joining large tables can be time-consuming. In this case, we can leverage B-AQP to answer the query approximately and achieve real-time responses.

Cloud-Native ML Analytics. While we can execute ML models with high throughput using hardware accelerators (e.g., H100x8), the throughput of cloud storage can be smaller due to limited bandwidth, blocking the query execution. We can use B-AQP to accelerate data loading.

Algorithm 1: Core Procedure of B-AQP Runtime.

Input : input query Q_i, target error e, target probability p, number of data pages N, data page access API MaterializePage, data process API Compute
Output : An approximate result R

```
R_p ← QueryProcess(Q_i, 100, PilotRewrite)
n ← DetermineSampleSize(Q_i, R_p)
R_f ← QueryProcess(Q_i, n, FinalRewrite)
return R_f
Function QueryProcess(Q_i, n, RewriteFn):
    S ← BlockSample(N, n)
    S ← MaterializePage(S)
    Q ← RewriterFn(Q_i, S)
    R ← Compute(Q, S)
    return R
```

3 B-AQP STATISTICAL ENGINE

In this section, we briefly introduce the statistical techniques based on block sampling that guarantee user-specified error targets (§3.1 and §3.2) and theoretically analyze the performance benefit (§3.3).

3.1 Block-Oriented Estimation

Block sampling introduces dependence among data records within the same page. The analytical confidence interval based on CLT assumes independently distributed data, which is not true for block sampling. We analyze the influence of such data dependence and propose block-oriented estimation to address it.

Failure of the naive approach. Consider a simple query that computes an `AVG` over a dataset with N pages, where each page P_i has K records: $\{X_{i,1}, \ldots, X_{i,K}\}$. Suppose we obtain a random sample of n pages. The estimated `AVG` aggregate is computed as $\hat{\mu} = \frac{1}{nK} \sum_{i=1}^{n} \sum_{j=1}^{K} X_{i,k}$. We can obtain the a 95% confidence interval: $\left[\hat{\mu} - z_{0.975} \cdot \sqrt{Var[\mu]}, \quad \hat{\mu} + z_{0.975} \cdot \sqrt{Var[\mu]}\right]$, where $z_{0.975}$ is the 97.5th percentile of the normal distribution. Due to the data dependence, we need to consider covariance terms in $Var[\mu]$. That is $Var[\mu] = \frac{1}{n \cdot K} Var[X_{i,j}] + \sum_{i=1}^{n} \sum_{j_1 \neq j_2} Cov[X_{i,j_1}, X_{i,j_2}]$. The covariance can be arbitrarily large, leading to invalid confidence intervals if we naively use the variance of sampled records for $Var[\mu]$.

Our Solution. We mitigate data dependence by computing the estimation at the level of data pages. Suppose S_i indicates the sum of page i, which is identically and independently distributed. The `AVG` aggregate can be equivalently estimated as $\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} \frac{S_i}{K}$, while the sampling variance is not only an unbiased estimator of

$Var[\mu]$ but also has an exponentially decaying tail if S_i is bounded [10]. When the page size K_i varies, the block-oriented estimator can be expressed as $\frac{N}{n}\sum_{i=1}^{n} S_i$ for SUM, $\frac{N}{n}\sum_{i=1}^{n} K_i$ for COUNT, and as the ratio between SUM and COUNT for AVG.

Complex Queries. Real-world queries are often more complex than simply computing aggregates over datasets. They may incorporate various relational operations such as selection and join. To address those complex operations, we can analyze the query with block sampling applied to the last derived table while pushing down the sampling procedure to the data loading phase during query execution. Similar to the proof of rejection sampling, we can prove that pushing down block sampling through projection, selection, and grouping does not affect the statistical distribution of aggregates. For the join operation, we can prove the validity of the pushdown if block sampling is executed on one of the join tables.

3.2 Pilot Sampling

Given a statistically valid way to compute confidence intervals for queries with block sampling, it remains challenging to output approximate results that meet the error specifications of users. To address this, we use pilot sampling, which issues a small sampling query to gather block-level statistics and calculate the sufficient sample size or sample rate.

Specifically, we find that the width of the confidence interval, which guarantees the target error, is a function of sample size. For example, the confidence interval width for SUM is calculated as $2 \cdot z_{0.975} \cdot N \cdot \sqrt{Var[S]}/\sqrt{n}$ where N is the total number of pages and S is the sum of a page. To ensure a specific confidence interval width, we use pilot sampling to estimate an upper bound for $Var[S]$ and compute the required sample size n. Similar procedures of pilot sampling can be generalized to other linear aggregates.

3.3 Theoretical Analysis

Although we can achieve a priori error guarantees for block sampling, it may require processing more data than uniform sampling for the same error targets. However, we argue that this is not a major concern for the overall cost. We theoretically analyze the benefit of using block sampling over uniform sampling.

Theorem 3.1. *Suppose a dataset $\{X_{i,j}|i = 1, \ldots, N; j = 1, \ldots, K\}$ consists of N pages, each containing K records. Let c_1 and c_2 represent the cost of accessing and processing a page, respectively. Given a target error e, we can obtain the following lower bound for the cost ratio.*

$$\frac{Cost(uniform)}{Cost(block)} \geq \frac{N}{n}\left(1-\left(1-\frac{m}{NK}\right)^K\right)\frac{c_1}{c_1+Kc_2} + \left(1-\frac{\mathbb{E}[Var[X_{i,j}|i]]}{Var[X]}\right)^{-1}\frac{c_2}{c_1+Kc_2} \quad (1)$$

where n and m is the required sample size to achieve target error e using block sampling and uniform sampling, respectively.

Proof. Given the layout of the dataset, we estimate the expected number of page accesses in uniform sampling, where Z_i is a random variable indicating whether P_i is accessed

$$\mathbb{E}\left[\sum_{i=1}^{N} Z_i\right] = N\cdot(1-\mathbb{P}\left[P_1 \text{ is not accessed}\right]) \geq N\cdot\left(1-\left(1-\frac{m}{NK}\right)^K\right)$$

Given the requirement of achieving the same error rate, we have

$$\frac{m}{n} = \frac{Var[X_{i,j}]}{Var[\mathbb{E}[X_{i,j}]|i]} = \frac{Var[X_{i,j}]}{Var[X_{i,j}] - \mathbb{E}[Var[X_{i,j}|i]]}$$

Finally, we can calculate the cost ratio as

$$\frac{Cost(uniform)}{Cost(block)} = \frac{\mathbb{E}\left[\sum_{i=1}^{N} Z_i\right]\cdot c_1 + m\cdot c_2}{n\cdot c_1 + n\cdot K\cdot c_2}$$

Applying the estimation of $\mathbb{E}\left[\sum_{i=1}^{N} Z_i\right]$, we obtain the final result. □

Based on Theorem 3.1, we find two scenarios where block sampling can save costs compared to uniform sampling. First, when the page size is large and the data access is significantly more expensive than data processing, both $1-(1-m/NK)^K$ and $c_1/(c_1+Kc_2)$ are close to 1. In this case, the cost ratio is at the order of $O(N/n)$, indicating a significant saving by block sampling. Second, when the variance of each page is close to the variance of the dataset, $\mathbb{E}[Var[X_{i,j}|i]]/Var[X_{i,j}]$ is close to 1. Hence, the second term of the ratio can be large, indicating cost savings by block sampling.

4 PRELIMINARY EVALUATION

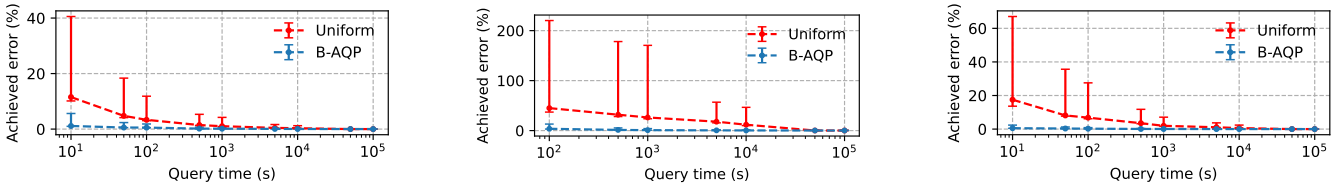
We developed a prototype of B-AQP based on an analytical database management system: DuckDB, and a cloud data system: Amazon S3. We focus on demonstrating three benefits of B-AQP: error guarantees, query accelerations, and error reductions.

Settings. We consider workloads with structured and unstructured data. For structured data, we evaluated B-AQP on the widely used 5,000-scale TPC-H workload. We executed query 12, which involves complex selection, grouping, and a two-way join. We conduct experiments on CloudLab r6525 nodes, each equipped with 256 GB RAM, 1.6 TB NVMe SSD, and two 32-core AMD 7543 CPUs.

For unstructured data, we evaluated B-AQP on twitter [8] and amazon [12] datasets. We scaled both datasets to 1 TB and executed simple aggregation queries with predicates on the output of ML models, simulating real-world tasks, such as sentiment analysis [4] and entity resolution [21]. We conduct experiments with 8 H100 GPUs, 200 GB RAM, a 48-core CPU, and a Gigabit NIC.

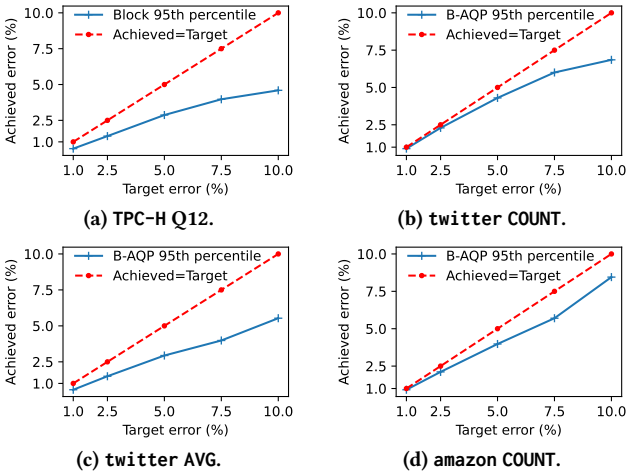
B-AQP Guarantees Errors. To evaluate whether B-AQP achieves a priori error guarantees, we executed B-AQP on all workloads with a 95% confidence level and different target errors. For each target error, we repeated experiments 100 times and report the 95th percentile errors. Figure 4 presents the achieved errors of B-AQP. As shown, the 95th percentile errors of B-AQP are consistently lower than the target errors, as indicated by red dashed lines. This demonstrates that B-AQP achieves error guarantees for both structured and unstructured data. We notice that the achieved error of B-AQP can be significantly lower than the target errors by up to 134%, indicating that the sample size estimation of B-AQP may not be optimal for specific workloads. We call for future work to better minimize the sample size for in B-AQP.

B-AQP Accelerates Query Processing. To evaluate the speedups that B-AQP can achieve, we compare B-AQP to exact queries and uniform sampling. We executed each method on all workloads across 1-10% target errors. For each target error, we repeated experiments 100 times and report the average query time in Figure



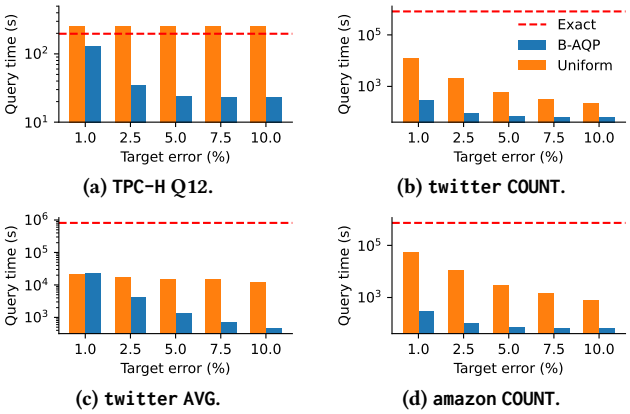
(a) `twitter COUNT`. (b) `twitter AVG`. (c) `amazon COUNT`.

Figure 3: B-AQP reduces errors by up to 41%, compared to uniform sampling. Error bars present the 5th and 95th quantiles.



(a) `TPC-H Q12`. (b) `twitter COUNT`. (c) `twitter AVG`. (d) `amazon COUNT`.

Figure 4: B-AQP achieves targeted errors for all workloads.



(a) `TPC-H Q12`. (b) `twitter COUNT`. (c) `twitter AVG`. (d) `amazon COUNT`.

Figure 5: B-AQP accelerates uniform sampling by up to 185× and exact queries by up to 12,412×. Query time is in log scale.

5. As shown, B-AQP achieves similar or faster query processing compared to uniform sampling, with 0.98-185× speedups. Compared to exact queries, B-AQP consistently achieves faster query processing, with speedups of up to 2,821× at 1% error and 12,412× at 10% error. A case study on `twitter AVG` with a 5% error reveals that for uniform sampling, data loading and processing on GPUs take 98% and 2% of query time, partly because the throughput of small language models (e.g., Bert) on H100x8 can be as high as 71K samples/s [18]. B-AQP achieves improvements by significantly reducing data loading time while keeping processing time similar.

B-AQP Achieves Smaller Errors. To evaluate the error performance of B-AQP given time budgets, we executed B-AQP and uniform sampling on all workloads across five different time budgets. For each time budget, we repeated experiments 100 times and report the median, 5th percentile, and 95th percentile achieved errors in Figure 3. We only show the error performance on unstructured data, as uniform sampling over structured data did not complete processing in the given time budget. As shown, B-AQP consistently achieves smaller errors compared to uniform sampling, with an improvement of up to 41.3%. B-AQP can achieve an error less than 5% within 500 seconds, significantly reducing latency.

5 RELATED WORK

Online AQP. Generating samples at query time, online AQP aims to accelerate query processing without prior knowledge of workloads [13, 14, 16, 20]. Different from offline AQP methods that maintain offline samples tailored for specific workloads [3, 17], online AQP eliminates the need for maintenance and assumptions on workloads. Previous work has developed various sampling algorithms [13, 14, 16, 20] to analyze estimation errors or minimize sample size. Unlike block sampling, they operate at the record level, which is inefficient in data systems that use pages as the access unit.

Block Sampling. Block-level sampling has long been recognized as a more efficient scheme than record-level sampling for query processing [6, 7, 9, 11, 19]. Prior work has investigated the `COUNT` estimator [11] and variance estimator [6] with block sampling. Researchers have developed various techniques to improve the error performance of block sampling [9, 19]. However, these methods do not allow users to specify error requirements.

6 CONCLUSION

In this work, we propose B-AQP, a novel AQP framework based on block sampling, which significantly improves data loading efficiency and provides a priori error guarantees. We develop a novel statistical engine to achieve those targets while outlining challenges and opportunities for future improvements. Our evaluation across various workloads and data systems reveals that B-AQP achieves error guarantees consistently and accelerates queries by up to four orders of magnitude compared to exact queries and up to 185× compared to uniform sampling.

REFERENCES

[1] Databricks Accelerators 1. [n. d.]. *Real-Time Point-of-Sale Analytics*. https://www.databricks.com/solutions/accelerators/real-time-point-of-sale-analytics Accessed: 2024-07-10.

[2] Databricks Accelerators 2. [n. d.]. *Automating Product Review Summarization With LLMs*. https://www.databricks.com/solutions/accelerators/automating-product-review-summarization-with-large-language-models Accessed: 2024-07-10.

[3] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*.

[4] Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves. 2020. TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification. In *EMNLP Findings*.

[5] CL Philip Chen and Chun-Yang Zhang. 2014. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information sciences* 275 (2014), 314–347.

[6] Xingguang Chen, Fangyuan Zhang, and Sibo Wang. 2022. Efficient Approximate Algorithms for Empirical Variance with Hashed Block Sampling. In *SIGKDD*.

[7] Xiang Ci and Xiaofeng Meng. 2015. An efficient block sampling strategy for online aggregation in the cloud. In *Web-Age Information Management: 16th International Conference*. Springer.

[8] Jonathan Falvey. 2023. ChatGPT Tweets. https://huggingface.co/datasets/deberain/ChatGPT-Tweets

[9] Peter J Haas and Christian König. 2004. A bi-level bernoulli scheme for database sampling. In *SIGMOD*.

[10] Wassily Hoeffding. 1994. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding* (1994), 409–426.

[11] Wen-Chi Hou and Gultekin Ozsoyoglu. 1991. Statistical estimators for aggregate relational algebra queries. *ACM Transactions on Database Systems (TODS)* (1991).

[12] Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. 2024. Bridging Language and Items for Retrieval and Recommendation. *arXiv:2403.03952* (2024).

[13] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*.

[14] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Accelerating approximate aggregation queries with expensive predicates. *PVLDB* (2021).

[15] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive data analytics on cold data using serverless cloud infrastructure. In *SIGMOD*.

[16] Supriya Nirkhiwale, Alin Dobra, and Christopher Jermaine. 2013. A sampling algebra for aggregate estimation. *PVLDB* (2013).

[17] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. Verdictdb: Universalizing approximate query processing. In *SIGMOD*.

[18] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[19] Kexin Rong, Yao Lu, Peter Bailis, Srikanth Kandula, and Philip Levis. 2020. Approximate partition selection for big-data workloads using summary statistics. *PVLDB* (2020).

[20] Matthew Russo, Tatsunori Hashimoto, Daniel Kang, Yi Sun, and Matei Zaharia. 2023. Accelerating Aggregation Queries on Unstructured Streams of Data. *PVLDB* (2023).

[21] Spacy. 2024. *EntityRecognizer*. https://spacy.io/api/entityrecognizer Accessed: 2024-07-10.

[22] Xiangyao Yu, Matt Youill, Matthew Woicik, Abdurrahman Ghanem, Marco Serafini, Ashraf Aboulnaga, and Michael Stonebraker. 2020. PushdownDB: Accelerating a DBMS using S3 computation. In *ICDE*.