



OmniSketch: Multi-dimensional update stream analytics with arbitrary predicates

Wieger R. Punter¹ · Odysseas Papapetrou¹ · Minos Garofalakis²

Received: 24 August 2025 / Revised: 28 November 2025 / Accepted: 24 December 2025 / Published online: 10 January 2026
© The Author(s) 2026

Abstract

A key need in different disciplines is to perform analytics over fast-paced data streams, similar in nature to the traditional OLAP analytics in relational databases - i.e., with aggregates and selection predicates. Storing unbounded streams, however, is not a realistic, or desired approach due to the high storage requirements, and the delays introduced when storing massive data. Accordingly, many synopses/sketches have been proposed that can summarize the stream in small memory (usually sufficiently small to be stored in RAM), such that aggregate queries can be efficiently approximated, without storing the full stream. However, past synopses predominantly focus on summarizing single-attribute streams, and cannot handle selection predicates and constraints on arbitrary subsets of multiple attributes efficiently. In this work, we propose OmniSketch, the first sketch that scales to fast-paced and complex data streams (with many attributes), and supports count aggregates with predicates on multiple attributes, dynamically chosen at query time. OmniSketch supports streams containing both inserts and deletes, under the bounded deletes streaming model. OmniSketch offers probabilistic guarantees, a favorable space-accuracy tradeoff, and a worst-case logarithmic complexity for updating and for query execution. We demonstrate experimentally with both real and synthetic data that OmniSketch outperforms the state-of-the-art and can approximate complex ad-hoc queries within the configured accuracy guarantees, with small memory requirements.

Keywords Multi-dimensional Stream Analytics · Sketching · Bounded deletes model

1 Introduction

Aggregates and selection predicates constitute the workhorse of data analytics, and are implemented in all databases. Accordingly, indexing and storage techniques have been implemented to answer such queries efficiently, even over big data. However, analytics on *streaming* data raises new challenges which call for novel approaches. In particular, due to the large size of contemporary streams – most of them are, in fact, unbounded – storing the full stream such

that it can be processed later for answering ad hoc queries becomes impractical. Second, in typical streaming applications, such as monitoring of network traffic data or finance data, extremely fast query response times (in the order of milliseconds) are required to support reactive applications. For example, in network monitoring for early DoS detection, a delay of a couple of seconds when answering a query for up-to-date network statistics may provide sufficient time to an attacker for compromising a corporate network. Third, the high stream update rate (millions of updates per second) that is typical in contemporary streams presents a key obstacle for using traditional data structures to speed-up the queries, e.g., by maintaining different types of indices.

The go-to techniques to handle streaming analytics are *sketches*: compact data structures that summarize the stream, typically with a fixed space complexity. These structures can be updated and queried in constant or polylogarithmic time, with theoretical error guarantees, and they typically work by relying on randomly chosen hash functions to transform the stream into a small-space summary [10]. The key idea is that the user knows the (set of) interesting queries for the user,

✉ Wieger R. Punter
w.r.punter@tue.nl

Odysseas Papapetrou
o.papapetrou@tue.nl

Minos Garofalakis
minos@athenarc.gr

¹ Eindhoven University of Technology, Eindhoven, The Netherlands

² ATHENA Research Center, Marousi, Greece & Technical University of Crete, Chania, Greece

and initializes the necessary sketches for addressing these queries before observing the stream. Sketches sacrifice some accuracy for a drastic improvement in space complexity, high throughput, and extremely fast querying. Most sketches are also backed by rigorous theoretical analysis, which allow the user to control the space/accuracy trade-off, with probabilistic accuracy guarantees. Owing to their simplicity and high performance, sketches have been widely adopted across distinct domains (both research and industrial applications), to meet diverse requirements. For example, Bloom filters [3] and their variants are frequently used for supporting membership queries on large sets, the celebrated AMS sketches [1] are used for computing L2 norms and similarities over distributions, HyperLogLog sketches [12] are frequently used for estimating distinct counts in streaming multisets, and Count-Min sketches [11] are the de facto data structure for approximating stream distributions.

The key limitation of sketches, however, is that each sketch can be used to address only one, or at best, a few, query templates. A query template refers to a *predefined* structure for aggregate queries over a data stream, characterized by a *fixed set of predicate attributes* while allowing variable predicate values. For example, consider the use case of network monitoring, which is crucial for ensuring the reliability, security and performance of network systems. Here, the Count-Min sketch is often used to maintain frequency statistics. A standard IPv4 packet header defines at least 13 attributes, including version, header length, total length, source and destination address, protocol. As a running example throughout this work, we consider a simplified IPv4 header stream of Figure 1, which contains just 4 attributes, and a sign (Δ) that denotes whether the update is an insert (+1) or delete (-1). To estimate queries adhering to query template `SELECT COUNT(*) FROM STREAM WHERE ipSrc=?` (i.e., number of packets per IP address), a Count-Min sketch using the source IP address as the hash key is required (see Section 2.2). Monitoring the number of packets received per destination IP (`ipDest`) can be done with another query template – thus an additional sketch that hashes on `ipDest`. To monitor the number of packets exchanged between any two IPs, we need to maintain yet another sketch, that uses the concatenation of the source-destination IP addresses as key.

In exploratory data analysis, the analyst requires the flexibility to answer *any* frequency query on the IPv4 stream, *on any possible combination of predicates*. However, the number of Count-Min sketches to support any potential query template grows exponentially with the number of attributes. In the toy example of Fig. 1, already $(2^4 - 1)$ sketches will need to be maintained, whereas in a more realistic scenario for IPv4 packets (with all 13 attributes), we would need to maintain $O(2^{13})$ sketches. This requirement stems from the way the sketches are constructed and queried: insert-

ipSrc	ipDest	totalLen	dscp	Δ
131.1.2.1	23.11.1.2	40	0	+1
147.3.4.7	147.3.4.8	48	32	+1
147.3.4.7	147.3.4.8	56	32	+1
131.1.2.1	23.11.1.2	40	0	-1
147.3.4.8	147.3.4.7	40	32	+1
...	...	...	...	...

Fig. 1 : A (simplified) stream, used as the running example. A sample query with multiple predicates on this stream may be, e.g., `SELECT COUNT(*) FROM stream WHERE ipSrc=131.2.2.1 AND ipDest=142.1.4.7 AND totalLen>40 AND dscp=0`

ing a tuple in the sketch is done by hashing together the same subset of attributes that will be used as query predicates later. Generalizing to arbitrary scenarios, estimating frequency distribution for all possible combinations of p predicates requires the maintenance of $2^p - 1$ sketches. This is clearly infeasible, because of both the involved space complexity and the strict efficiency constraints that arise in the context of data streams.

In this work, we focus on supporting frequency queries over multi-dimensional streams that follow the turnstile model, which allows both insertions and deletions (i.e., negative updates). Specifically, we consider the specialization of the model, known as the *bounded deletes model* [16], where the ratio of deletions to insertions is bounded to enable additional algorithmic optimizations. The model is essential for accurately representing many scenarios. For example, deletes are needed to represent transactions in a store, where clients may purchase but also return a product. In high-churn adhoc networks, the model reflects devices dynamically joining and leaving the network. Similarly, when treating database contents as a stream of updates (with inserts and deletes) – as is common in query optimization – it is important to account for deletions that remove records from the database.

Our contributions We propose OmniSketch, the first sketch that allows compact and efficient summarization of (possibly unbounded) fast-paced streams under the bounded deletes model, and subsequent efficient execution of general queries, decided at query time, with accuracy guarantees. The sketch combines the compactness of sketches, which is necessary for reducing the memory constraints, with the generality of sampling, which is key for supporting general queries, on predicates that are dynamically decided at query time. In a nutshell, OmniSketch consists of one compact sketch per attribute – a 2d matrix, similar to Count-Min sketch. Each cell in these matrices contains a fixed-size sample of the inserted record-ids that hash in the cell, plus some additional statistics to support fast inserts and deletions. At

query time, arbitrary conjunctions of predicates are supported by looking-up each predicate individually on the corresponding attribute sketch, and computing the intersection of the results (the retrieved record-ids), thereby translating the conjunction of predicates into a set-intersection problem.

Unlike previous work, OmniSketch offers computational complexity (for both updates and queries) that scales *linearly* with the number of attributes – instead of exponentially – rendering it the only viable, general-purpose solution, to date, for summarizing fast-paced streams with many attributes in small space and with efficient querying. OmniSketch comes with formal error guarantees and an automated initialization algorithm that builds on the theoretical analysis to fully utilize the available memory.

We evaluate OmniSketch experimentally on both real and synthetically-generated streams, and compare it with the state-of-the-art competitors: Hydra [21] and adaptive Sample and Hold with signed updates [8]. Our experiments confirm that OmniSketch is the only viable option for summarizing complex fast-paced streams, and comes with a favorable accuracy/efficiency trade-off. In contrast, Hydra becomes extremely slow when summarizing streams with five or more attributes, and therefore fails to effectively summarize fast-paced streams. While adaptive Sample and Hold scales significantly better than Hydra, it still suffers from limited throughput and high query latency – both critical factors in high-frequency and latency-sensitive applications.

Roadmap The remainder of the paper is structured as follows. In Section 2 we present the preliminaries and discuss related work. In Section 3 we present OmniSketch for the cash register model (i.e., without deletes) and analyse its theoretical properties. Section 4 presents an extension of OmniSketch (coined OmniSketch_B) that supports streams containing deletes. Section 5 summarizes our experimental results, and in Section 6 we conclude the paper and discuss our future plans.

2 Preliminaries and related work

This work is an extended version of [24], which considered support for only insert-only streams. In this new work, we add support for streams under the bounded deletes model. We start by discussing the bounded deletes model, before moving on to two key related techniques, Count-Min and Min-wise hashing, which are central to our OmniSketch design. Then, we discuss other related work from the literature.

2.1 The bounded deletes model and supported queries

We adopt the bounded deletes streaming model of Jayaram and Woodruff [16], which allows for both positive updates (insertions of new records) and negative updates (deletions of records that were previously inserted). This model additionally supports bounding the number of negative updates to not exceed a percentage of the stream insertions at any given time, thereby allowing for novel and simpler streaming algorithms with a lower space and computational complexity [16], e.g., Space-Saving $\pm$ [29], KLL $\pm$ [30].

Formally, the input data is a stream of records $\mathcal{S} = \{r^{(1)}, r^{(2)}, \dots, r^{(N)}\}$, e.g., similar to the records of our running example of Figure 1. Let $\mathcal{A} = \{a_1, a_2, \dots, a_{|\mathcal{A}|}\}$ denote the stream attributes (in the running example, ipSrc, ipDest, totalLen, dscp). Each record comes with a record id (rid), denoted by a_0 – or, $r_0^{(i)}$ for a record $r^{(i)}$. If no record ids exist, such unique rids can be easily constructed at ingestion time (e.g., by using a primary key, a counter, or a timestamp). Records also contain an insertion/deletion flag, denoted as $r_{\Delta}^{(i)}$ for record $r^{(i)}$. This takes a value from $\{+1, -1\}$, with $r_{\Delta}^{(i)} = +1$ denoting an insertion, and $r_{\Delta}^{(i)} = -1$ denoting a delete of a previously inserted record. With $\mathcal{I}$ we denote all insertions from $\mathcal{S}$, i.e., $\mathcal{I} := \{r \in \mathcal{S} | r_{\Delta} = +1\}$ and with $\mathcal{D}$ the set of all stream deletions, i.e., $\mathcal{D} := \{r \in \mathcal{S} | r_{\Delta} = -1\}$. The bounded deletes model also allows us to bound the number of deletes, as follows: $|\mathcal{D}| \leq (1 - \frac{1}{\alpha})|\mathcal{I}|$, for a domain-specific value α set by the user. Setting $\alpha = 1$ leads to the cash register model, i.e., no deletes allowed, whereas with $\alpha = \infty$ we allow all previously inserted records to be also deleted. The value of α can be typically easily estimated, or upper-bounded. For example, in a store, one could expect the returns to never surpass, say, 10% of the sales, leading to $\alpha = \frac{1}{1-0.1} \approx 1.11$.

At any point in time, the residual stream $\mathcal{R}$ contains all records that were inserted but not yet deleted, i.e., $\mathcal{R} = \mathcal{I} - \mathcal{D}$. This also implies that, in the absence of deletes (i.e., for streams under the cash register model, which only allows insertions), $\mathcal{R} = \mathcal{I}$.

Supported queries Let $\mathcal{S}$ denote the stream arrivals up until query time, and $\mathcal{R} = \{r^{(1)}, r^{(2)}, \dots\}$ denote the residual stream. The query q is a count query, comprising a conjunction of selection predicates for any subset $\{a_i, a_j, a_k, \dots\}$ of $\mathcal{A}$:

$$f(q) := \text{Count} (r \in \mathcal{R} \mid r \text{ satisfies } q = q_i \wedge q_j \wedge q_k \dots)$$

where each predicate q_i can be a disjunction of range and equality predicates on an individual attribute a_i .

2.2 Count-Min sketch

Proposed in 2005, the Count-Min sketch [11] quickly became the de facto sketch for approximate frequency distributions of streaming data. A Count-Min sketch CM is a 2-dimensional array of width w and depth d , accompanied by d pairwise-independent hash functions $\{h_1, h_2, \dots, h_d\}$ that map the input to the range $\{1, \dots, w\}$. Let $CM[j, k]$ denote the counter at cell (j, k) in the sketch's 2-dimensional array. A record r (e.g., an IP address) is added once to the sketch by increasing the count at $CM[j, h_j(r)]$, for all $j \in \{1 \dots d\}$. The frequency of any query item q in the stream can be estimated by finding the corresponding cells $CM[j, h_j(q)]$ per row j , and returning the minimum count, i.e., $\hat{f}(q) = \min_{j \in \{1 \dots d\}}(CM[j, h_j(q)])$. Due to hash collisions (items other than q that fall in the same counters) $\hat{f}(q)$ is potentially an overestimate of the true frequency $f(q)$. By setting $w = \lceil e/\epsilon \rceil$ and $d = \lceil \ln(1/\delta) \rceil$, we have that $\hat{f}(q) - f(q) \leq \epsilon N$ with probability $\geq 1 - \delta$, where N is the length of the stream.

It is straightforward to use Count-Min sketches for summarizing frequency distributions of pairs, triples, or arbitrary subsets of attributes. For instance, in the running example of Fig. 1, one could summarize the distribution of $\langle \text{ipSrc}, \text{ipDest} \rangle$ by simply using the concatenation of the two attributes as a *composite* key in all hashes – both at insertion and querying. A single Count-Min sketch is then required, per attribute combination. This approach however becomes prohibitively expensive when we need to summarize many such combinations. Even worse, in the typical exploratory data analytics scenario where the interesting combinations are decided at query time – based on the intermediary results – the number of sketches that need to be constructed grows exponentially to the number of attributes. Hence, this naive approach does not scale.

2.3 Min-wise hashing

OmniSketch relies on min-wise hashing to estimate the cardinality of the intersection of multiple sets. The key idea behind all min-wise hashing (or MinHash) schemes is to hash all items in each set using one or more hash functions, and to keep only the B items with the smallest hash values per hash function as samples of each set. The size of the intersection of these samples for the different sets can then be scaled to estimate the cardinality of the intersection of the sets.

The min-wise hashing algorithm that we will be using in this work estimates the sets intersection cardinality as follows [23]: Let $R_1, R_2, \dots, R_p$ denote the p sets for which we want to estimate the cardinality of their intersection. First, we construct the summary S_i for each set R_i by hashing each item with a global hash function $g(\cdot) \rightarrow \{0, 1\}^b$,

and retaining the B smallest hash values, with b set to at least $\lceil \log(4B^{2.5}/\delta) \rceil$. The cardinality of the intersection of p sets $|R_\cap| = |R_1 \cap \dots \cap R_p|$ can then be estimated from these summaries as $\hat{R}_\cap = |\bigcap_{i=1}^p S_i| * n_{\max}/B$ with $n_{\max} = \max_{i=1}^p |R_i|$. This estimate comes with (ϵ, δ) guarantees when $\hat{R}_\cap \geq 3n_{\max} \log(2p/\delta)/(B\epsilon^2)$. When the above condition fails, a weaker bound can be shown: $0 \leq |R_\cap| \leq 4n_{\max} \log(2p/\delta)/(B\epsilon^2)$, with probability $1 - \delta\sqrt{B}$.

We chose min-wise hashing over other sampling methods, since it has been shown to perform equally well or outperform other sampling methods (including [18, 19]), and has space complexity that nearly matches the theoretical lower bound for the problem [23]. Furthermore, the chosen algorithm works with streaming data, as the samples can be maintained incrementally.

Min-wise hashing assumes a cash register stream and cannot handle deletions. In particular, deletes can deplete the sample when elements are removed, thus corrupting the similarity estimates. A possible workaround would be to recompute the sketch after each deletion. However, this would require: (a) storing the whole set, and, (b) processing the whole set after some deletes, in order to maintain the k smallest signatures. Both steps are prohibitively expensive, especially when focusing on fast-paced, unbounded data streams. Section 4.1 describes how we address these issues. Because min-wise hashing is widely used, our extended variant that handles deletions is of independent interest and can be directly applied to diverse applications.

2.4 Other related work

Hydra The state-of-the-art sub-population sketch for multi-dimensional data streams is Hydra [21]. At the outer layer, Hydra consists of a Count-Min sketch of width w and depth d , accompanied with d pairwise-independent hash functions that map the input domain to $[1 \dots w]$. Each cell in this sketch contains a nested universal sketch [20], for keeping detailed statistics. When a record of p attributes is read, it is hashed in the Hydra sketch as follows. First, all $2^p - 1$ possible combinations of the record's attributes are computed. Each of these combinations defines a sub-population where this record belongs. Each combination is then hashed with the d hash functions to find the corresponding cells in the outer sketch (one cell per row), and the combination is finally added in the contained universal sketches. For query execution, the query predicates are combined to create a key, which is then hashed using the same d hash functions for finding the corresponding cells in the outer sketch. We then query the nested universal sketch using the same key, to estimate the frequency.

In a thorough experimental evaluation, Hydra is shown to outperform other approaches, and provides interactive query latency. However, the sketch has two critical downsides that make it a non-viable option for streams with many attributes. First, recall that each record is added $2^p - 1$ times in the sketch, with different keys – once per possible combination. As we will demonstrate later, this exponential complexity becomes a challenge already for a modest number of indexed attributes, i.e., $p = 4$, in terms of time complexity, causing problems when summarizing fast-paced streams. Second, because of this sharp increase (exponential with p) of the number of additions to the sketch, the approximation error also rapidly increases. Our experimental results show that while Hydra demonstrates excellent performance for small p values, its error (as well as update time) increase exponentially with p .

Adaptive Sample and Hold with signed updates Adaptive Sample and Hold (aSH) algorithm with signed updates [8] represents the state-of-the-art for stream sampling with deletions. The algorithm dynamically adjusts the sampling threshold, raising it when the sample fills with positive updates, and lowering it as negative updates free up space. In this way it maintains a representative sample of bounded size over a stream with insertions and deletions.

The algorithm keeps a sample S of size K . For each sampled record s_i included in the sample, it keeps two additional values: (a) a counter c_i , initialized to Δ , and, (b) τ_i , the effective sampling threshold for the record, initialized to 0. New records are added to the sample until it reaches size K . Then, two passes are made over the sample. In the first pass, T_i , which is a randomized function on τ_i , is recomputed, and the minimum threshold $\tau^* = \min_{s_i \in S}(T_i)$ is found. In the second pass, the record with $T_i = \tau^*$ is ejected from the sample, while the thresholds of surviving records are adjusted according to another randomized function. Frequency queries are executed by iterating over the sample and summing the values of both c_i and τ_i for all records in the sample that satisfy all predicates. This is shown to be an unbiased estimator [8].

Adaptive Sample and Hold has two downsides, both related to efficiency. First, queries are slow, as they require a full iteration over the whole sample, which may be in the order of tens to hundreds of Megabytes – especially for high-dimensional streams. Second, updates are also slow, as the ejection step in the sampling procedure causes two passes over the collected samples. To improve the throughput of updates, the authors propose use of a buffer that allows batching the updates and reducing the number of ejection procedures. This way, instead of ejecting samples one by one after each update (with each ejection requiring two passes over the sample), all updates contained in the buffer are processed at the same time, allowing the ejection of multiple items after adjusting the sampling thresholds. This buffer

drastically improves the ingestion throughput of aSH. However, as we demonstrate in Section 5, even with the buffer, aSH efficiency for both queries and updates is still substantially lower than the throughput of OmniSketch_B, which is a significant limitation for a streaming algorithm.

Similarity estimation on dynamic sets Min-wise hashing [23] and other variants from the MinHash scheme family [4, 9, 18, 19] are efficient and well understood in the insertion-only streaming model, but they do not support deletions, as explained in Section 2.3. This observation has motivated research specifically targeted to support deletions. The state-of-the-art is the l -buffered k -MinHash sketch [7], which, similar to our α -min-wise sampling, exploits buffers to mitigate deletes. In particular, l -buffered k -MinHash scheme maintains one buffer per hash, where it keeps some extra items with a small hash value, to recover the k samples in case of deletes without reverting to the full dataset. Their buffered scheme yields attractive amortized update costs, low storage and querying complexity and outperforms earlier competitors [5]. However, the algorithm still needs to maintain the full data, in order to recover from the case that the buffer is insufficient. Furthermore, the algorithm can lead to incorrect estimates when a deleted minimum should have been replaced by an element that appeared earlier but was discarded at the time because it exceeded the then-active threshold. Ignoring such events can make the sketches inconsistent with the sketch computed from the residual set.

Summarizing, despite the extensive work on MinHash and derivatives, there is still insufficient work on similarity estimation on dynamic sets with deletes, with the state-of-the-art requiring to store the full dataset, and failing to properly account for deletes. Our algorithm, presented in Section 4.1 extends the min-wise hashing approach and its analysis to support bounded dynamic updates. It avoids reliance on external storage for recovery and eliminates the inconsistency described above, preserving the desirable min-wise hashing guarantees.

Our algorithm is therefore also useful outside OmniSketch_B, as a replacement of min-wise hashing [23] and of other MinHash variants in diverse applications such as similarity search [7], and neighborhood queries on graphs [2, 28]. Exploring the application of α -min-wise sampling on such use cases is promising future work, but outside the scope of the present paper.

3 OmniSketch for cash register streams

We now present OmniSketch for cash register streams, a sketch that allows efficient frequency estimation of queries with predicates over fast-paced insert-only streams. We gradually construct the final sketch, in three successive steps.

Initially, we describe an extension of the standard Count-Min sketch, termed $\mathbb{S}0$, that maintains additional data in the cells. This additional information is leveraged for estimating cardinalities involving predicates using a straightforward generalization of the Count-Min estimator. Then, keeping the $\mathbb{S}0$ sketch structure unchanged, we present an improved estimation algorithm and the corresponding theoretical analysis to tighten the error bounds, at no extra complexity. We refer to the new sketch estimator as $\mathbb{S}0_\cap$, to distinguish from the earlier estimator (termed $\mathbb{S}0_{\min}$). Then, we present the cash register OmniSketch ($\mathbb{S}1$) which merges $\mathbb{S}0_\cap$ with a sampling technique to guarantee sublinear space complexity.

Unlike previous works, all sketches allow very efficient updates (with logarithmic complexity) and queries, and can therefore easily handle fast-paced streams and a high throughput on queries.

All three sketches are based on a common architecture: each comprises a collection of sub-sketches, with one sketch assigned to each searchable attribute, i.e., an attribute for which predicate support is desired. We refer to the sub-sketches as attribute sketches, and denote them by $CM_1, CM_2, \dots, CM_{|\mathcal{A}|}$, where $\mathcal{A}$ corresponds to the set of searchable attributes. Each CM_i is an array of size $w \times d$, accompanied with d pairwise-independent hash functions $h_1^i(\cdot), h_2^i(\cdot), \dots, h_d^i(\cdot)$ (similar to Count-Min sketches). However, unlike traditional CM-sketches, each cell in CM_i contains a list of record ids (rids), or their fingerprints. Rids are placed/hashed in each sketch based on the record value on the corresponding attribute. During query execution, the attribute sketches corresponding to the query's predicates are queried, and the record ids retrieved by these are intersected, to compute an estimate of the answer. Figure 2 illustrates the general sketch architecture, for a sample dataset with searchable attributes: $\langle \text{ipSrc}, \text{ipDest}, \text{totalLen}, \text{dscp} \rangle$. The difference between the three sketches relates to: (a) the way the record ids are sampled and stored in the attributes sketches, and (b) the theory backing the estimators which affects the tightness of the bounds.

In the ensuing discussion, we use $\mathcal{A} = \{a_1, a_2, \dots, a_{|\mathcal{A}|}\}$ to denote the set of searchable attributes, q to denote a query that contains p predicates, and q_i to denote the predicate for attribute $a_i \in \mathcal{A}$. For ease of presentation, unless otherwise mentioned, we assume that each predicate q_i is an equality predicate on attribute a_i with a constant value v_{q_i} ; that is, $q_i := (a_i = v_{q_i})$. The domain of each attribute a_i is denoted by $\mathcal{D}(a_i)$.

Without loss of generality, we assume that q contains all attributes of $\mathcal{A}$, i.e., $p = |\mathcal{A}|$ (if some of the attributes are not contained in the query, we simply ignore the corresponding sketches during query execution); in general, $p \leq |\mathcal{A}|$. We summarize frequently used notation in Table 1.

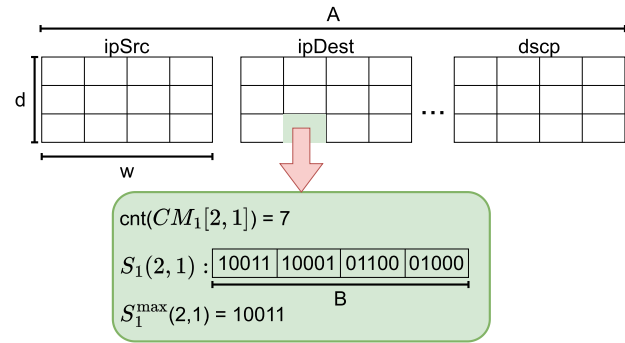


Fig. 2 : OmniSketch. The green-shaded box illustrates the contents of a cell in $\mathbb{S}1$

3.1 Sketch $\mathbb{S}0$: Count-Min with rid-sets for estimating queries with predicates

Initialization At initialization time, we choose uniformly at random $|\mathcal{A}| \times d$ pairwise-independent hash functions $h_1^1(\cdot), h_2^1(\cdot), \dots, h_d^1(\cdot)$, with $h_i^j(\cdot) : \mathcal{D}(a_i) \rightarrow \{1, \dots, w\}$. We also construct $|\mathcal{A}|$ arrays $CM_1, CM_2, \dots, CM_{|\mathcal{A}|}$, of size $w \times d$, and initialize each of their cells to contain an empty linked list. In order to get (ϵ, δ) -guarantees on the estimate, we set $w = 1 + \lceil e/\epsilon \rceil = \Theta(1/\epsilon)$ and $d = \lceil \ln(1/\delta) \rceil$.

Insertion A new record r needs to be added to all $|\mathcal{A}|$ sketches. We use r_i to denote the value of record r on attribute a_i , and r_0 to denote its unique record id (rid). For each searchable attribute a_i and for each row $j = \{1, \dots, d\}$, we add r_0 to all linked lists at positions $CM_i[j, h_i^j(r_i)]$.

Note that, as defined, the $\mathbb{S}0$ structure is not strictly a “sketch” since, by storing complete rid-sets, it requires space that is linear in the stream length N . The only summarization $\mathbb{S}0$ performs is by “blurring” individual attribute values through hashing into Count-Min buckets. Still, it provides a conceptually useful first step towards our final OmniSketch solution.

The $\mathbb{S}0_{\min}$ Estimator. Let $f(q)$ denote the number of records satisfying all predicates in q . Following the conventional Count-Min estimation procedure [11], we can compute an estimate $\hat{f}(q)$ as follows: for each row $j = \{1, \dots, d\}$, we compute the size of the intersection of all records stored in cells $CM_1[j, h_1^j(v_{q_1})], CM_2[j, h_2^j(v_{q_2})], \dots, CM_p[j, h_p^j(v_{q_p})]$. We repeat this for the d rows, and return the minimum value as an estimate. Formally:

$$\hat{f}(q) = \min_{j=\{1 \dots d\}} \left| \bigcap_{q_i \in q} CM_i[j, h_i^j(v_{q_i})] \right| \quad (1)$$

Intuitively, the intersection will contain the rids of all records r that satisfy the query, since these will always hash at the same cell as the query predicate, for all predicates,

Table 1 Frequently used notation

Notation	Description
$\mathcal{I}$	Insertion stream.
$\mathcal{D}$	Deletion stream.
N	Stream length.
α	Bounded delete ratio $\alpha < 1/(1 - \mathcal{D} / \mathcal{I})$
$f(q)$	Number of records satisfying query q
d	Sketch depth (number of rows)
w	Sketch width (number of columns)
B	Maximum sample size per cell
CM_i	Sketch for attribute i .
$CM_i[j, k]$	Cell at row j , column k , of sketch CM_i .
$h_i^j(\cdot)$	Hash function for attribute sketch, row j of attribute i .
$R(CM_i[j, k])$	Set of records hashed to cell $CM_i[j, k]$.
$S_i(j, k)$	Set of records in the sample of cell $CM_i[j, k]$.
$C(q)$	Set of cells $CM_i[j, h^j(v_{q_i})]$ accessed to answer the query.
$R_\cap$	Set of records hashed to all cells in $C(q)$.
$n_{\max}$	Max. number of records hashed to any cell $CM_i[j, k] \in C(q)$.
β	Buffer factor for α -min-wise sample.
K	Guaranteed queryable sample size of α -min-wise sample.

However, the intersection may also contain some false positives, i.e., records that were hashed in $CM_i[j, h_i^j(v_{q_i})]$ due to one or more random collisions. By taking the minimum intersection size across all d rows, we try to minimize the effect of such false positives.

Error bounds The following lemma provides probabilistic guarantees for the estimator of Eqn. 1.

Lemma 1 Let $\hat{f}(q)$ be the estimate provided by Equation 1 on sketches constructed with $d = \lceil \ln(\frac{1}{\delta}) \rceil$, $w = 1 + \lceil \frac{e}{\epsilon} \rceil$. For query q with p predicates:

$$Pr\left(|\hat{f}(q) - f(q)| \leq \epsilon N\right) \geq 1 - \delta \quad (2)$$

Proof Consider row $j \in \{1 \dots d\}$. Let us define an indicator variable $I_{y,j}$, which takes the value of 1 for all records y satisfying the following condition:

$$\forall_{q_i \in q} [h_i^j(v_{q_i}) = h_i^j(y_i)] \wedge \exists_{q_i \in q} [v_{q_i} \neq y_i]$$

Informally, the described condition is true for all false positive records, i.e., records that are contained in the intersection but do not fully satisfy the query. Also, variable $X^j = \sum_{y=1}^{N-f(q)} I_{y,j}$ is a counter of these false positives for row j .

Recall that the estimator for row j is as follows: $\hat{f}^j(q) = |R_\cap^j|$, where $R_\cap^j$ denotes the intersection of the cells at position $h_i^j(v_{q_i})$, for $i = \{1, \dots, p\}$. We need to prove that: (a) all records that satisfy all predicates will be contained in $R_\cap^j$,

i.e., there will be no false negatives, and (b) the total number of false positives X is upper-bounded w.h.p.

For (a), notice that for any record y satisfying $y_i = v_{q_i}$ for $i = \{1 \dots p\}$, we will have $y_i = v_{q_i} \Rightarrow h_i^j(y_i) = h_i^j(v_{q_i})$. Therefore, all records satisfying all predicates will be included in $R_\cap^j$.

For (b), a record will be a false positive and increase the value of X^j by 1 if for all attributes y_i for which $y_i \neq v_{q_i}$, we have $h_i^j(y_i) = h_i^j(v_{q_i})$. Let X_k^j denote the number of records contained in $R_\cap^j$ that satisfy exactly $p-k$ predicates, i.e., they do not satisfy k of the p predicates, but they are retrieved by the algorithm due to hash collisions. Then, $X^j = \sum_{k=1}^p X_k^j$.

By construction of the hash functions, a record y with $y_i \neq v_{q_i}$ will have $h_i^j(y_i) = h_i^j(v_{q_i})$ with probability $1/w$. Since the hash functions are pairwise-independent, the probability that a record differing from q in k attributes hashes in the same cells is $1/w^k$. Therefore, $E[X_k^j] \leq (N - f(q))/w^k$, and $E[X^j] = \sum_{k=1}^p E[X_k^j] \leq (N - f(q)) * (1/w + 1/w^2 + \dots + 1/w^p) = (N - f(q)) * (1 - (1/w)^p) / (w - 1)$. By setting $w = 1 + \lceil e/\epsilon \rceil$ we get $E[X] = (N - f(q)) / e * (1 - \epsilon^p) / \epsilon \leq \epsilon(N - f(q)) / e \leq \epsilon N / e$.

Notice that we have d rows, each with different hash functions. By Markov's inequality, we have

$$\begin{aligned} Pr[X \geq \epsilon(N - f(q))] &= Pr[\forall_{j \in \{1 \dots d\}} \cdot \hat{f}^j(q) > f(q) + \epsilon N] \\ &= Pr[\forall_{j \in \{1 \dots d\}} \cdot X^j > eE[X^j]] \\ &< e^{-d} \end{aligned}$$

where $f(q)$ denotes the true answer and X_j denotes the false positives at row r , i.e., $X_j = \hat{f}^j(q) - f(q)$. The final bound follows by setting $d = \lceil \ln(1/\delta) \rceil$. $\square$

$\mathbb{S}0_\cap$: An Improved Estimator. The $\mathbb{S}0_{\min}$ estimator is based on the conventional estimation logic in the standard Count-Min sketch [11]; that is, it produces an estimate per row and takes the minimum across all the d row estimates. Similar to standard Count-Min estimation, it is easy to see that $\mathbb{S}0_{\min}$ can only overestimate the true count due to hash bucket collisions (false positives). However, compared to the standard Count-Min, each bucket in $\mathbb{S}0$ contains much more detailed information that can be exploited to provide much tighter estimates (i.e., with less false positives). The key observation here is that, by construction of $\mathbb{S}0$, each record that satisfies the full query will end up in the cells of *all* d rows for this query and for all p predicates, whereas false positives are expected to only end up in a few of the rows. Thus, without modifying the sketch construction or space/time complexity, we can get a tighter estimator by removing the min operation across rows and simply intersecting the rid-sets returned by all d rows. We refer to this tighter $\mathbb{S}0$ estimator as $\mathbb{S}0_\cap$, and formally define it as follows:

$$\hat{f}(q) = \left| \bigcap_{i=\{1 \dots p\}, j=\{1 \dots d\}} CM_i[j, h_i^j(v_{q_i})] \right| \quad (3)$$

Clearly, this new $\mathbb{S}0_\cap$ estimator can also only overestimate the true count due to hash collisions – at the same time, it guarantees fewer (or the same) false positives since it is always less than or equal to the $\mathbb{S}0_{\min}$ estimate in Eqn. 1. We now demonstrate the stronger error guarantees of $\mathbb{S}0_\cap$, proving that it allows us to bound the error by $\epsilon^d(N - f(q))$ instead of $\epsilon(N - f(q))$, without changing the space complexity of the sketch.

Lemma 2 *Let $\hat{f}(q)$ be the estimate provided by Equation 3 on sketches constructed with $d = \lceil \ln(1/\delta) \rceil$, $w = 1 + \lceil \frac{\epsilon}{\epsilon} \rceil$. For query q with p predicates:*

$$Pr\left(|\hat{f}(q) - f(q)| \leq \epsilon^d(N - f(q)) \leq \epsilon^d N\right) \geq 1 - \delta$$

Proof The proof is similar to the proof for Lemma 1.

We define an indicator variable I_y , which takes the value of 1 for all records y satisfying the following condition:

$$\forall_{q_i \in q} [\forall_{1 \leq j \leq d} [h_i^j(v_{q_i}) = h_i^j(y_i)] \wedge \exists_{q_i \in q} [v_{q_i} \neq y_i]]$$

Informally, I_y becomes 1 for all false positive records, i.e., the records that hash to the same cells as the query predicate value, at *all* d rows and at *all* p predicates. Also, $X = \sum_{y=1}^{N-f(q)} I_y$ is a counter of the total number of false positive records.

We will prove that (a) all records that satisfy all predicates will be contained in $\bigcap_{j=1}^d R_\cap^j$, i.e., there will be no false negatives, and, (b) the number of false positives X is upper-bounded w.h.p..

For (a), notice that for any record y satisfying $y_i = v_{q_i}$ for $i = \{1 \dots p\}$, we have $y_i = v_{q_i} \Rightarrow \forall_{1 \leq j \leq d} [h_i^j(y_i) = h_i^j(v_{q_i})]$. Therefore, all records satisfying all predicates will be included in the intersection of all $R_\cap^j$, denoted as $R_\cap$.

For (b), a record y will increase the value of X if for all attributes y_i for which $y_i \neq v_{q_i}$, and for all rows $j \in \{1 \dots d\}$, it collides with $h_i^j(v_{q_i})$, i.e., $h_i^j(y_i) = h_i^j(v_{q_i})$. The probability of this collision at a single row j is $Pr[h_i^j(y_i) = h_i^j(v_{q_i})] = 1/w$. Furthermore, by independence of the hash functions across sketch rows, the probability that a record collides with the query at all d rows j will be $1/w^d$.

Let X_k denote the number of records from $R_\cap$ that satisfy *exactly* $p - k$ predicates. Then, $E[X_k] \leq (N - f(q))/w^{kd}$, and $E[X] = \sum_{k=1}^p E[X_k] \leq (N - f(q)) \sum_{k=1}^p 1/w^{kd} = (N - f(q)) \frac{1 - w^{-d(p+1)}}{w^d - 1} < (N - f(q))/(w^d - 1)$. By setting $w = 1 + \lceil \epsilon/\epsilon \rceil$, we get $E[X] \leq (N - f(q))/((1 + \epsilon/\epsilon)^d - 1) \leq \epsilon^d(N - f(q))/\epsilon^d$.

The following bound follows directly from Markov inequality:

$$Pr[X \geq \epsilon^d(N - f(q))] = Pr[X \geq \epsilon^d E[X]] \leq e^{-d}$$

Then, by setting $d = \lceil \ln(1/\delta) \rceil$ we get our final bound, $Pr[X \geq \epsilon^d(N - f(q))] \leq \delta \Rightarrow Pr[X \geq \epsilon^d N] \leq \delta$ $\square$

3.2 Sketch $\mathbb{S}1$ (OmniSketch): Achieving sublinear space through sampling

Sketch $\mathbb{S}0$ stores the record ids of all records, resulting in a space complexity of $O(d \times |\mathcal{A}| \times N)$, which is not viable for large data streams. Our full-fledged cash register OmniSketch (also denoted by $\mathbb{S}1$ in what follows), achieves space complexity strictly sublinear in N by taking and maintaining *samples* of rids in the cells of the attribute sketches, using a min-wise hashing algorithm [23]. We now describe our OmniSketch solution, assuming that two key parameters (the maximum sample size per cell B and the range of the sampling hash function $[0, 2^b - 1]$) are already set. We explain how these values are determined later in this section.

Similar to $\mathbb{S}0$, $\mathbb{S}1$ is composed of a collection of $|\mathcal{A}|$ attributes sketches. Furthermore, $\mathbb{S}1$ incorporates a hash function $g : \mathcal{D}(r_0) \rightarrow \{0, 1\}^b$, which maps each record id to a bit string of length b , with b set to at least $\lceil \log(4B^{2.5}/\delta) \rceil$. Function g is necessary for min-wise hashing (Section 2). The $|\mathcal{A}|$ attributes sketches $CM_1, CM_2, \dots, CM_{|\mathcal{A}|}$ are all

Algorithm 1 Adding a record r to the sketch

```

1: Input: record  $r$ 
2: for  $i \in \{1 \dots |\mathcal{A}|\}$  do
3:   for  $j \in \{1 \dots d\}$  do
4:      $k \leftarrow h_i^j(r_i)$ 
5:      $\text{cnt}(CM_i[j, k])++$ 
6:     if  $|S_i(j, k)| < B$  then
7:       Add  $g(r_0)$  to  $S_i(j, k)$ 
8:       Recompute  $S_i^{\max}(j, k)$ 
9:     if  $|S_i(j, k)| = B$  then
10:      if  $g(r_0) < S_i^{\max}(j, k)$  then
11:        Remove  $S_i^{\max}(j, k)$  from  $S_i(j, k)$ 
12:        Add  $g(r_0)$  to  $S_i(j, k)$ 
13:        Recompute  $S_i^{\max}(j, k)$ 

```

of the same size $w \times d$. Each sketch cell $CM_i[j, k]$ contains: (a) the count of all items that were hashed in this cell, denoted as $\text{cnt}(CM_i[j, k])$, (b) a min-wise sample $S_i(j, k)$ of maximum size B , of the rid hash values $g(r_0)$ of all records r that were hashed in this cell, and, (c) the maximum hash value of all items contained in $S_i(j, k)$, denoted as $S_i^{\max}(j, k)$. Notice that $\text{cnt}(CM_i[j, k])$ also includes items that were hashed in $CM_i[j, k]$ but did not end up in the sample, and it is expected to be much greater than B in a populated sketch. Fig. 2 (the green-shaded box) shows an example of a populated cell's contents, with $B = 4$ and $b = 5$.

Initialization The user chooses the desired error guarantees (ϵ, δ) , with $\epsilon < 0.25$. Let (ϵ_1, δ_1) and (ϵ_2, δ_2) correspond to the (ϵ, δ) configurations of the min-wise hashing algorithm and the attribute sketch respectively, and $\epsilon_1 = \epsilon$, $\epsilon_2 = (\epsilon/(1 + \epsilon))^{1/d}$, and $\delta_1 = \delta_2 = \delta/2$. We initialize all attribute sketches, by choosing w and d similar to $\mathbb{S}_0$. Each cell in these sketches is initialized with an empty sample $S_i(j, k)$, and with $\text{cnt}(CM_i[j, k]) = 0$ and $S_i^{\max}(j, k) = \infty$.

Insertion To add a new record r to the sketch, we first locate all cells across the $|\mathcal{A}|$ sketches that correspond to the values of r (lines 2-4, Alg. 1). These are the cells $C(r) = \{CM_i[j, h_i^j(r_i)] : i = \{1, \dots, |\mathcal{A}|\}, j = \{1, \dots, d\}\}$. For each of these cells $CM_i[j, k] \in C(r)$, we increase $\text{cnt}(CM_i[j, k])$ by one (line 5). Then, we examine whether $g(r_0)$ should be added to the sample of the cell (lines 6-10), as follows. If the sample contains less than B items, then $g(r_0)$ is added to it. If, on the other hand, $|S_i(j, k)| = B$ and $g(r_0) < S_i^{\max}(j, k)$ (i.e., the sample is full, but the new record's id has a smaller hash value g compared to another record from the sample), we remove $S_i^{\max}(j, k)$ from the sample to make space for $g(r_0)$. Finally, we add $g(r_0)$ to $S_i(j, k)$, and recompute $S_i^{\max}(j, k)$.¹ The complexity of inserting an element is $O(|\mathcal{A}| \times d \times \log(B))$.

¹ To speed-up the computation of $S_i^{\max}(j, k)$, as well as the estimation process, the samples $S_i(j, k)$ are maintained in a red-black tree. Therefore, re-computation of $S_i^{\max}(j, k)$ takes (amortized) constant time.

OmniSketch Query Estimation. Let q denote the input query. As with $\mathbb{S}_0$, we assume that q contains all $\mathcal{A}$ attributes, i.e., $p = |\mathcal{A}|$; if an attribute is not contained in q , the estimator simply ignores the corresponding attribute sketch. Each query attribute value v_{q_i} is hashed using the d hash functions of the corresponding sketch CM_i , which leads to d cells per sketch. Let $C(q)$ denote the set of cells across all sketches that are accessed to answer the query, i.e., $C(q) = \{CM_i[j, h^j(v_{q_i})] : i = \{1 \dots p\}, j = \{1 \dots d\}\}$, and $n_{\max} = \max_{CM_i[j, k] \in C(q)} \text{cnt}(CM_i[j, k])$ be the maximum count of these cells. Following the reasoning of our improved $\mathbb{S}_0$ estimator, let

$$S_{\cap} = \bigcap_{i=\{1 \dots p\}, j=\{1 \dots d\}} S_i(j, h^j(v_{q_i}))$$

denote the intersection of all samples stored in all cells in $C(q)$. Our OmniSketch estimator $\hat{f}(q)$ is computed as follows:

$$\hat{f}(q) = \frac{n_{\max}}{B} \times |S_{\cap}| \quad (4)$$

Intuitively, this estimator computes the cardinality of the intersection of all samples in $C(q)$ and scales it up by a factor of $\frac{n_{\max}}{B}$, to account for the sampling. Based on the analysis of min-wise hashing [23], our estimator comes with a sanity bound of $\frac{3n_{\max} \log(4pd\sqrt{B/\delta})}{B\epsilon^2}$ to cover the case of insufficient samples in the intersection.

Efficient computation of the intersection Computing $|S_{\cap}|$ requires a multi-way join over all samples that participate in the query. For large values of B , a naive hash-based computation of this intersection can take hundreds of milliseconds. To speed up query execution, we exploit the fact that the samples at each cell are already stored in a red-black tree, which allows for sorted iteration and search with complexity logarithmic in B . Our code is an extension of the sort-merge join for multi-way joins. Starting from the first cell, we get the first sample (the one with the minimum hash value), and execute a lookup on all other cells to check whether this hash value is contained in their samples. As soon as we find a cell that does not contain this sample, we get the smallest sample from that cell with a hash value greater than the one that failed, and resume our search from this value (no other value in between the failed and this value could be part of the join). An interesting observation, which also becomes obvious in our experimental results (see Section 5), is that this algorithm typically becomes more efficient when answering queries with more predicates, because it allows for larger steps. For example, if one of the cells contains very few samples, this cell will lead to skipping many candidate records at all other cells. This can bring the complexity of query

execution from $O(p * d * B * \log(B))$ down to effectively $O(B \log(B))$.

Derivation of error bounds The following theorem provides the error guarantees for our OmniSketch estimator. The nature of the guarantee, whether it depends on ϵN or on $n_{\max}/\epsilon^2$, is determined by the number of witnesses of the intersection ($|S_{\cap}|$). It is important to note that in any realistic setting, $n_{\max} \ll N$.

Theorem 1 Consider an OmniSketch with $\epsilon_1 = \epsilon$, $\epsilon_2 = (\epsilon/(1+\epsilon))^{1/d}$, and $\delta_1 = \delta_2 = \delta/2$. If $|S_{\cap}| < \frac{3 \log(4pd\sqrt{B}/\delta)}{\epsilon^2}$, setting $\hat{f}(q) = \frac{2n_{\max} \log(4pd\sqrt{B}/\delta)}{B\epsilon^2}$ satisfies $|f(q) - \hat{f}(q)| < \frac{2n_{\max} \log(4pd\sqrt{B}/\delta)}{B\epsilon^2}$ with probability at least $1 - \delta/2$. Otherwise, setting $\hat{f}(q) = \frac{n_{\max}}{B} \times |S_{\cap}|$ satisfies $|f(q) - \hat{f}(q)| \leq \epsilon N$ with probability at least $1 - \delta$.

Proof sketch. $f(q)$ has two sources of errors: (a) underestimation or overestimation due to min-wise sampling, and, (b) (one-sided) overestimation due to hash collisions in the outer Count-Min structure. We first provide a bound on the sampling error using the analysis in [23], which is oblivious to hash collisions, and then integrate the error due to hash collisions, of Lemma 2. The full proof of Theorem 1 can be found in [24] (Theorem 3.3).

The following corollary simplifies the estimation procedure, by always using the second estimator.

Corollary 1 Let $\hat{f}(q) = \frac{n_{\max}}{B} \times |S_{\cap}|$. If $|S_{\cap}| < \frac{3n_{\max} \log(4pd\sqrt{B}/\delta)}{B\epsilon^2}$, then $|f(q) - \hat{f}(q)| \leq \frac{4n_{\max} \log(4pd\sqrt{B}/\delta)}{B\epsilon^2}$ with probability at least $1 - \delta/2$. Otherwise, the same estimator satisfies $|f(q) - \hat{f}(q)| \leq \epsilon N$ with probability at least $1 - \delta$.

The corollary follows directly from the theorem, by noticing that when $|S_{\cap}| < \frac{3n_{\max} \log(4pd\sqrt{B}/\delta)}{B\epsilon^2}$, then $f(q) \in \{0, \frac{4n_{\max} \log(4pd\sqrt{B}/\delta)}{B\epsilon^2}\}$ with high probability, and $\hat{f}(q) \in \{0, \frac{4n_{\max} \log(4pd\sqrt{B}/\delta)}{B\epsilon^2}\}$. Therefore, with high probability, the estimator is at most $\frac{4n_{\max} \log(4pd\sqrt{B}/\delta)}{B\epsilon^2}$ away of $f(q)$.

Complexity The space complexity of $\mathbb{S}1$ is $O(w \times d \times B \times b \times |\mathcal{A}|)$. Computational complexity for query execution is $O(p \times d \times B \times \log(B))$, and for insertions is $O(|\mathcal{A}| \times d \times \log(B)/t)$, where t is a small buffer for ingestions that is flushed when full or at query time and the last logarithm is for maintaining an ordered set of samples, which speeds up insertions and queries.

Configuring the sketch The user sets the available memory M , and the values of ϵ and δ . The values of w and d are computed as follows: $w = 1 + \lceil e * ((\epsilon + 1)/\epsilon)^{1/d} \rceil = \Theta((1/\epsilon)^{1/d})$ and $d = \lceil \ln(2/\delta) \rceil$. In order to not exceed the memory quota M (in bits), the user chooses the maximum B

that satisfies $M \geq w * d * |\mathcal{A}| * (32 + B * (\lceil \log(4B^{2.5}/\delta) \rceil + 1))$.²

3.3 Extensions

Support for range queries All proposed sketches support range queries on numeric attributes, as well as combinations of range and point predicates, e.g., counting the number of records with $integer(112.1.4.1) \leq ipSrc \leq integer(112.1.255.255)$, $integer(202.21.1.1) \leq ipDest \leq integer(202.22.255.255)$, and $0 \leq dscp \leq 16$. This is achieved by indexing and querying dyadic ranges, similar to the technique used in Count-min sketches [11]. In particular, the ranges-enabled OmniSketch contains $\log(|\mathcal{D}(a_i)|)$ internal attributes sketches for each attribute a_i , denoted as $CM_{(i,0)}, CM_{(i,1)}, \dots, CM_{(i,\log(|\mathcal{D}(a_i)|)-1)}$. Each internal sketch $CM_{(i,k)}$ keeps the frequency statistics for dyadic ranges of length 2^k . Therefore, $CM_{(i,0)}$ stores statistics for points, $CM_{(i,1)}$ stores statistics for ranges of size 2, and so on.

A record r is indexed into the sketch as follows. For each numerical attribute r_i , and for $k = 0$ to $\log(|\mathcal{D}(a_i)|) - 1$, we find all ranges of the form $[x * 2^k + 1, (x + 1)2^k]$ that contain r_i , where $x \in \mathbb{Z}^*$. For each range of size 2^k , we add x to $CM_{(i,k)}$. Query execution follows a similar process. At query time, any range predicate is broken to its canonical cover – all sub-ranges that follow the above form. Then, for each row of the sketch, we query all sub-ranges and merge the retrieved samples, effectively constructing a single sample that covers the complete query range. The remaining querying process remains identical to point queries.

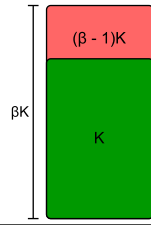
Maintenance of the internal range sketches increases the space and time complexity of the sketch by a factor of $\log(|\mathcal{D}(a_i)|)$. The approximation error depends on the number of dyadic ranges contained in the canonical cover of the query, which is at most $2 \log(|\mathcal{D}(a_i)|)$, as in [14]. Therefore, the total error is at most $2\epsilon \log(|\mathcal{D}(a_i)|)N$.

4 OmniSketch_B for bounded deletes

In this section, we describe OmniSketch_B, a novel sketch that extends OmniSketch to support streams with deletions. OmniSketch_B is designed for the widely-used bounded deletes model, where the deletions-to-insertions ratio can be bounded from above, e.g., the number of deletions does not exceed 90% of the number of insertions in the stream. OmniSketch_B leverages this bound, parameterized via α (see Section 2), to optimize its space requirements and increase

² To the best of our knowledge, there is no closed-form solution of this formula. However, the solution can be efficiently approximated with a numerical computation toolkit, e.g., using the bisection method.

Fig. 3 : Sample of size βK . Red indicates buffer, Green indicates queryable sample



Algorithm 2 Insert record (r)

```

1:  $t \leftarrow t + 1$ 
2:  $h \leftarrow g(r_0)$ 
3: if  $c \leq \beta K$  then
4:   Insert  $h$  into  $S$ 
5:    $c \leftarrow c + 1$ 
6:    $\tau \leftarrow \max(S)$ 
7: else if  $h < \tau$  then
8:    $\min_r \leftarrow \min(\min_r, \tau)$ 
9:   Remove  $\tau$  from  $S$ .
10:  Insert  $h$  into  $S$ .
11:   $\tau \leftarrow \max(S)$ 
12: else  $\min_r \leftarrow \min(\min_r, h)$ 

```

its throughput. On the theoretical side, the sketch relies on a novel sampling algorithm for maintaining correct min-wise samples in the presence of deletions. On the technical side, provisioning for deletions induces additional computational complexity, which is addressed with a revised data structure. Note that OmniSketch_B reduces to the insertion-only sketch $\mathcal{S}1$ when configured with a 0% deletions-to-insertions ratio.

We start by introducing α -min-wise sampling, a novel sampling algorithm designed for the bounded deletes model. Briefly, α -min-wise sampling guarantees correct maintenance of a min-wise sample of streams with deletions. We then establish the theoretical foundation linking the parameter α to the correctness guarantees of the sampling algorithm. Finally, we integrate α -min-wise sampling into OmniSketch to derive OmniSketch_B .

4.1 α -min-wise sampling

$\mathcal{S}1$, the sketch for insert-only streams, relies on the min-wise hashing algorithm [23] to estimate the cardinality of intersections across cells. However, standalone min-wise hashing fails to maintain a correct sample when deletions occur. Specifically, deleting a sampled record creates an empty slot. The optimal replacement – the record with the next smallest hash value among all records – may have already been observed in the stream but was discarded at the time for having a hash value above the threshold. α -min-wise sampling extends the standard min-wise hashing of [23] to address this issue. The key idea is straightforward: maintain a standard min-wise sample of size K as in [23], while storing additional backup samples to *replenish* slots vacated by deletions.

Figure 3 depicts how α -min-wise sampling works. To guarantee a min-wise sample of size K after deletions, the

Algorithm 3 Delete record (r)

```

1:  $t \leftarrow t - 1$ 
2:  $h \leftarrow g(r_0)$ 
3: if  $h > \tau$  then return
4: if  $h \in S$  then
5:   Remove  $h$  from  $S$ .
6:    $c \leftarrow c - 1$ 
7:    $\tau \leftarrow \max(S)$ 

```

algorithm maintains a larger min-wise sample of size βK , with $\beta > 1$, e.g., $\beta = 1.5$. The extra $(\beta - 1)K$ samples act as a buffer to anticipate deletions. At query time, the $K' \geq K$ records with the lowest hash values are used as the queryable sample.

In the following, let S denote the whole sample.

Initialization. The algorithm maintains four parameters in addition to the samples: (a) t , the total number of inserted records, including those not kept in the sample, (b) the hash value threshold τ , used to decide record inclusion in the sample, (c) c , the current sample size, and (d) the minimum rejected value $\min_r$, which keeps track of the smallest hash value of any record that was either ejected from the samples, or failed to get in. The intuition behind $\min_r$ is that it will enable us to find the subset of samples from S that are not affected by any of the deletions. Initially, $\tau = \min_r = \infty$, and $c = 0$.

Insertions. When a record r arrives, increment t and compute $g(r_0)$. Three cases follow (Alg. 2):

- $c \leq \beta K$ (lines 3-6): Insert $g(r_0)$, increment c , and set τ to the maximum hash signature in S .
- $c = \beta K$ and $g(r_0) < \tau$ (lines 7-11): If the current $\min_r$ is higher than τ (the maximum signature in S), set $\min_r$ to τ and evict τ from S . Insert $g(r_0)$ to S and update τ to the value of the maximum signature in S .
- $c = \beta K$ and $g(r_0) \geq \tau$ (line 12): Reject the record, and update $\min_r$ if necessary.

$\min_r$ tracks the minimum hash among all evicted or rejected items. This means that, at any point of time, we can guarantee that none of the evicted or rejected records has a hash value less than $\min_r$. We use this property at query time to derive the value of K' .

Deletions. For a deletion (Alg. 3), decrement t and compute $g(r_0)$. Then:

- $g(r_0) > \tau$ (line 3): The record is not in S . Do nothing.
- $g(r_0) \leq \tau$ and $g(r_0) \in S$ (lines 4-7): Remove $g(r_0)$ from S , decrement c and update τ to correspond to the value of the maximum signature in S .

- $g(r_0) \leq \tau$ and $g(r_0) \notin S$: This case happens when threshold τ was increased between insertion time and deletion time of r . Do nothing.

Query execution. Let $S_1, S_2, \dots, S_k$ be the samples to intersect, with c_i, t_i , and K'_i corresponding to the values of c, t and K' for each sample S_i . We first compute $S_{\cap} := \bigcap_{i=1}^k S_i$, $t_{\max} := \max_{i=1}^k t_i$, and $i_{\max} := \arg \max_{i=1}^k t_i$ (i.e., the id of the sample with the maximum t).

For each sample S_i , K'_i is computed by counting the number of samples in S_i with a hash value less than $\min_r$, i.e., $K'_i = |\{r \in S_i \mid g(r_0) < \min_r\}|$. Then, the estimate of the cardinality of the intersection of all streams is computed using the same min-wise hashing estimator used by $\mathbb{S}1$:

$$\hat{f} := \frac{t_{\max}}{K'_{i_{\max}}} |S_{\cap}| \quad (5)$$

Configuration of β As long as each sample S_i participating in the query contains at least $K'_i \geq K$ valid samples, the estimator of Eqn. 5 is the standard min-wise hashing estimator of [23], and comes with the same error guarantees. We will now consider the probability that there exists at least one sample S_i participating in the query, with $K'_i < K$. In this case, the estimator remains valid and its error probability must be interpreted using the smaller effective sample size K'_i rather than K . The experiments in Figure 22 show that in practice the error degrades gracefully in this case.

The value of β controls the ability of the sketch to replenish the K min-wise samples after deletions. At the same time, a high β leads to higher space complexity and slower updates. Therefore, we want to set β as small as possible, to reduce the space complexity of the algorithm, but also sufficiently large such that the number of deletions from the sample is not larger than the buffer size, with high probability.

Let $\mathcal{I}$ and $\mathcal{D}$ correspond to the sub-streams of all inserts and all deletions from our stream. We will model the problem as follows: the population size is $|\mathcal{I}|$, out of which βK will also end up in the sample. Since we are using the same estimator as standard min-wise hashing for K samples, the accuracy of our estimator will be compromised when the number of surviving samples K' after executing all $\mathcal{D}$ deletions becomes less than K . This happens if at least $(\beta - 1)K + 1$ records are removed from the sample. In the opposite case, we say that the queryable sample remains valid.

The following lemma allows us to adjust the probability that a queryable sample remains valid in the bounded deletes model by changing β . Intuitively, we first model the probability that a deletion removes one record from a sample of size βK , exploiting the fact that the hash function $g(\cdot)$ maps each record uniformly in the hashing domain. This probability follows the Hypergeometric distribution [25]. Therefore, the

probability that at least $(\beta - 1)K$ deletions succeed removing one record from the sample (different each time), is modeled by the cumulative Hypergeometric distribution function F . This allows us to use this distribution to set βK , the total number of samples such that the probability that the sample becomes invalid (hence the accuracy of the estimator is compromised) is less than a probability δ .

Lemma 3 Let $\alpha \geq 1$ and $0 < \delta < 1$. By setting β such that it satisfies

$$F\left((\beta - 1)K, |\mathcal{I}|, \beta K, (1 - 1/\alpha)|\mathcal{I}|\right) \geq 1 - \delta \quad (6)$$

the probability that queryable sample is valid is $\Pr[X \leq (\beta - 1)K] \geq 1 - \delta$.

Proof Let X be a random variable that denotes the number of deletions from the sample. As the deletions are drawn without replacement uniformly over the insert stream, X follows the Hypergeometric distribution [25], i.e.,

$$X \sim \text{Hypergeometric}(|\mathcal{I}|, \beta K, (1 - 1/\alpha)|\mathcal{I}|)$$

where $(1 - 1/\alpha)|\mathcal{I}|$ represents the maximum number of deletions in the stream under the bounded deletes model. For $X \leq (\beta - 1)K$ we know that the queryable samples is valid, since any removal from it is automatically replenished from the buffer sample, and the remaining queryable samples will be $K' \geq K$. We want to know the minimum value of β for which the queryable sample will be valid, i.e., we want to solve $\Pr[X \leq (\beta - 1)K] \geq 1 - \delta$ for β :

$$\begin{aligned} \Pr[X \leq (\beta - 1)K] &= F\left((\beta - 1)K, |\mathcal{I}|, \beta K, |\mathcal{D}|\right) \\ &\geq F\left((\beta - 1)K, |\mathcal{I}|, \beta K, (1 - \frac{1}{\alpha})|\mathcal{I}|\right) \\ &\geq 1 - \delta \Rightarrow \\ &F\left((\beta - 1)K, |\mathcal{I}|, \beta K, (1 - \frac{1}{\alpha})|\mathcal{I}|\right) \\ &\geq 1 - \delta \end{aligned}$$

□

For known values of K , $|\mathcal{I}|$ and α , Inequality 6 allows us to choose the smallest β ensuring that the probability that the sample is valid is at least $1 - \delta$, for arbitrary values of δ and α . We are not aware of a closed-form solution for solving Inequality 6 on β . However, using a numerical computation toolkit (e.g., SciPy [27]) we can compute the required β in a few milliseconds for the given parameters. Also notice that in the typical case, we do not know the exact value of $|\mathcal{I}|$. However, for $|\mathcal{I}| > \times 10^6$, the optimal β does not change

significantly. For example, with $K = 10,000$, $\alpha = 2$, and 15 samples to intersect, the computed β value is 2.07 ± 0.01 for all $|I| \in \{10^6, 10^7, 10^8, 10^9\}$. Therefore, we use a default value of $|I| = 10^6$ for computing the required β .

The above lemma helps us compute the required β for a given α , such that a single queryable sample remains valid. However, estimating the cardinality of the intersection between multiple streams requires a sample per stream, and all of them need to remain valid. The following theorem can be used for properly initializing the individual samples, such that all samples are valid.

Theorem 2 Consider set $\mathcal{S} = \{S_1, S_2, \dots, S_n\}$ of n α -min-wise samples, summarizing n streams, denoted with $R_1, R_2, \dots, R_n$. All streams satisfy the bounded deletes model, with parameters $\alpha_1, \alpha_2, \dots, \alpha_n$ respectively. Let $\alpha = \max_{i=1}^n \alpha_i$. Random variable X_i denotes the (unknown) number of deletes from sample S_i . By setting $\delta_s = 1 - \sqrt[n]{1 - \delta}$ and β_i such that it satisfies $\Pr[X_i \leq (\beta_i - 1)K] \geq 1 - \delta_s$ for each sample $S_i \in \mathcal{S}$ (according to Lemma 3), the probability that any sample used for the query is not valid (i.e., it contains less than K samples) is $\Pr[\exists_{S_i \in \mathcal{S}} [X_i > (\beta - 1)K]] \leq \delta$.

Proof Let X_i be the number of deletions from sample $S_i \in \mathcal{S}$ and Y_i be an indicator random variable that takes value 1 if $X_i > (\beta - 1)K$, and 0 otherwise. We want to compute the probability that for all i , $X_i = 0$, i.e., all queryable samples S_i used in the query are still valid. This probability is $\prod_{i=1}^n \Pr[Y_i = 0]$. For each stream R_i with a bounded deletes parameter α_i , we configure sample S_i with parameter β_i such that $\Pr[Y_i = 0] \geq 1 - \delta_s$. Then, $\prod_{i=1}^n \Pr[Y_i = 0] \geq \prod_{i=1}^n (1 - \delta_s) = (1 - \delta_s)^n = 1 - \delta$, for $\delta_s = 1 - \sqrt[n]{1 - \delta}$.

For practicality, we further simplify this process by assuming a common value of $\alpha = \max_{i=1}^n \alpha_i$ for all streams. Then, using Inequality 6 of Lemma 3, we can find the smallest common β for all samples for which each sample remains valid with probability at least $1 - \sqrt[n]{1 - \delta}$. $\square$

4.1.1 Algorithmic optimizations

The throughput and memory complexity of α -min-wise sampling depends on the data structure used for storing the set of samples S . The frequently executed operations on the sample include: (a) checking if a record is included in S , (b) adding/removing a record, (c) finding the maximum value contained in S , and, (d) computing $|S_1 \cap S_2 \cap \dots \cap S_i|$, the size of the intersection of i samples.

Red-black trees (or other types of self-balancing trees) allow for high efficiency in all operations – at most $O(\log(\beta K))$ cost for the first three operations, and $O(i * \beta K)$ for computing the size of the intersection, using, e.g., the leap-frog join algorithm [26], which is worst-case optimal. However, these data structures double the memory requirements for

maintaining the pointers. Hash maps are more memory-efficient, but they require cost linear to βK for finding the maximum value contained in S , which is necessary after every insertion or deletion.

In this work, we store S in a conventional array of size βK , denoted as S_{arr} , which we maintain sorted during insertions and deletions. Since the array is sorted, locating a record in the sample requires only a binary search of complexity $O(\log(\beta K))$. The intersection of two or more samples is also computed efficiently with leap-frog join. Furthermore, iterating over the samples, which is a key operation in leap-frog joins, is very efficient because the samples are stored in consecutive memory. A downside of this representation is the high cost of memory copy operations during insertions and deletions, which becomes important for the throughput of the algorithm, particularly when βK is large. We mitigate this issue by maintaining two small buffers, an *insertion buffer* $\mathcal{B}_i$ and a *deletion buffer* $\mathcal{B}_d$, each with capacity to store a few tens of updates. The role of these buffers is to allow batching of the memory copy operations, e.g., instead of doing one (potentially large) memory copy for each insert, we do $|\mathcal{B}_i|$ smaller ones for all inserts contained in $\mathcal{B}_i$, and similarly for the deletes.

More precisely, buffering works as follows. All updates (both inserts and deletes) with hash values less than τ are pushed in the two buffers instead of being executed directly in S_{arr} . These buffers are flushed in three cases: (a) before querying, such that S_{arr} is up-to-date, (b) when $\mathcal{B}_i$ is full, to make room for more updates, and, (c) whenever $|\mathcal{B}_d| = |\mathcal{B}_i|$, even if the buffers are not yet full. It can be shown that these three conditions are sufficient for the algorithm with buffers to simulate α -min-wise sampling without buffers during query execution. Intuitively, condition (a) maintains that the algorithm always uses an up-to-date sample for query execution, whereas condition (b) ensures that no data will be lost due to full buffers. Finally, condition (c) guarantees that the value of τ of the algorithm with buffers will always be greater than or equal to the threshold of the algorithm without buffers. This guarantees that any update – insertion or deletion – that may affect S will be stored in the corresponding buffers, and eventually flushed to S_{arr} , leading to a sample that is identical to the sample of the algorithm without the buffers.

The flushing algorithm works in the following sequence. First, each buffer is sorted in descending order. Then, we start from $\mathcal{B}_i$, applying all inserts using a reverse merge. Particularly, for each insertion $\mathcal{B}_i[x]$, we use binary search to find the position that it should be inserted in S_{arr} for maintaining the sorted order. Let this position be x' , i.e., $S_{arr}[x' - 1] < \mathcal{B}_i[x] < S_{arr}[x']$. Since $\mathcal{B}_i$ is sorted in descending order, we know that the remaining $|\mathcal{B}_i| - x - 1$ inserts following $\mathcal{B}_i[x]$ in $\mathcal{B}_i$ are smaller than $\mathcal{B}_i[x]$, and will need to be inserted before $S_{arr}[x']$. Therefore, we preempt-

tively shift all samples following $S_{arr}[x']$ by $|\mathcal{B}_i| - x - 1$ positions, to their final positions after all inserts, and then set $S_{arr}[x' + |\mathcal{B}_i| - x - 1]$ to $\mathcal{B}_i[x]$. When inserting the next sample, $\mathcal{B}_i[x + 1]$, we no longer need to shift the samples from $S_{arr}[x' + |\mathcal{B}_i| - x - 1]$ onward. As a result, each hash value in the samples needs to be moved at most once while flushing out the whole buffer. Importantly, while inserting new hash values in the sample, we do not delete items from S_{arr} . Assuming that both $\mathcal{B}_i$ and S_{arr} were full, we will need to make room in S_{arr} for inserting $|\mathcal{S}_i|$ items. However, to anticipate for the pending $|\mathcal{B}_d|$ deletes, the $|\mathcal{B}_i|$ hash values that would otherwise be evicted during the insertions are temporarily stored in a small array of size $|\mathcal{B}_d|$, such that they can be retrieved back after executing the deletes, if needed. Following, we execute all deletes contained in $\mathcal{B}_d$, using the same reverse merge approach.

4.2 OmniSketch_B

We will now present OmniSketch_B, which integrates α -min-wise sampling with OmniSketch.

The data structure and initialization As a data structure, OmniSketch_B is very similar to S1, with the only difference that it uses α -min-wise sampling to maintain the samples at each cell. At initialization, the user configures the number of attributes $|\mathcal{A}|$ and the desired error guarantees (ϵ, δ) , with $\epsilon < 0.25$. Let (ϵ_1, δ_1) and (ϵ_2, δ_2) correspond to the (ϵ, δ) configurations of the min-wise sampling algorithm and the attribute sketch respectively. Finally, δ_3 corresponds to the failure probability of α -min-wise sampling. These values are automatically set as follows: $\epsilon_1 = \epsilon$, $\epsilon_2 = (\epsilon/(1 + \epsilon))^{1/d}$, and $\delta_1 = \delta_2 = \delta_3 = \delta/3$. The value of β is also automatically computed such that $Pr[X_i \leq (\beta - 1)K] \geq (1 - \delta_3)^{1/(|\mathcal{A}|*d)}$, according to Theorem 2 for intersecting $|\mathcal{A}| * d$ samples.

Updates The update operations are similar to S1, but now all records are added on the insertion and deletion buffers of the α -min-wise samples of the cells, as explained in Section 4.1.

Query execution Queries are answered as follows. First, all relevant cells for the query are identified, similar to S1. We denote the set of α -min-wise samples contained in these cells with $C(q)$. For all samples $c \in C(q)$, their insertion and deletion buffers ($\mathcal{B}_i$ and $\mathcal{B}_d$) are flushed. Then, we intersect the samples contained in $C(q)$, and use the estimator of Eqn. 5 to reach to an estimate for the query answer. It is important to note here that this estimator utilizes all K' records guaranteed to be valid at each α -min-wise sample, and not only the first K .

We will now derive theoretical guarantees for the accuracy of the estimates of OmniSketch_B.³ We need to consider two error factors, ϵ_1 and ϵ_2 , and three failure probabilities, δ_1 , δ_2 , and δ_3 . Error ϵ_1 stems from the min-wise hashing estimation, whereas ϵ_2 is due to hash collisions in the sketch, i.e., when two or more attribute values hash to the same cell. Failure probability δ_1 is the probability that the error due to sampling surpasses ϵ_1 , and δ_2 is the probability that the error due to hash collisions in the sketch surpasses ϵ_2 . Finally, δ_3 is the probability that α -min-wise sampling fails to maintain a sufficient number of samples $K' \geq K$ in at least one of the samples used for the query. Notice that, in the absence of failures of α -min-wise sampling, i.e., for the case that K' is greater than or equal to K , Theorem 2 guarantees that the answers of α -min-wise sampling will be identical to the answers of min-wise hashing (for the same values of K') and the proof boils down to the proof of Theorem 1 for S1. Therefore, we now need to combine Theorems 1 and 2, to support $\delta_3 \geq 0$.

Lemma 4 follows from the two theorems and allows us to bound the estimation error of OmniSketch_B.

Lemma 4 Consider an OmniSketch_B with $\epsilon_1 = \epsilon$, $\epsilon_2 = (\epsilon/(1 + \epsilon))^{1/d}$, $\delta_1 = \delta_2 = \delta_3 = \delta/3$, and parameter β set such that it satisfies $Pr[X_i \leq (\beta - 1)K] \geq \frac{1}{|\mathcal{A}|*d} \sqrt{1 - \delta_3}$ for each sample $c_i \in C(q)$. If $|S_{\cap}| < \frac{3 \log(6 * pd \sqrt{K/\delta})}{K \epsilon^2}$, then setting $\hat{f}(q) = \frac{2n_{\max} \log(6pd \sqrt{K/\delta})}{K \epsilon^2}$ satisfies $|f(q) - \hat{f}(q)| < \frac{2n_{\max} \log(6pd \sqrt{K/\delta})}{K \epsilon^2}$ with probability at least $1 - 2\delta/3$. Otherwise, setting $\hat{f}(q) = \frac{n_{\max}}{K} \times |S_{\cap}|$ satisfies $|f(q) - \hat{f}(q)| \leq \epsilon(|I| - |D|) \leq \epsilon|N|$ with probability at least $1 - \delta$, where I and D is the sub-streams containing the insertions and deletions, respectively.

Proof For executing a query of p predicates, we use α -min-wise samples to compute the intersection of $|C(q)| = p * d$. We distinguish two cases:

Case 1 There exists at least one sample in $C(q)$ that is invalid, i.e., X , the number of deletes from the sample is greater than or equal to $(\beta - 1)K$. In this case, the error bounds of α -min-wise sampling are not guaranteed. This happens with probability δ_3 .

Case 2 All samples contained in $C(q)$ are valid, despite the deletions. This happens with probability $1 - \delta_3$. In this case, each sample contains at least K valid records, and the results of Theorem 1 (both the error ϵ and the failure probability $1 - \delta_1 - \delta_2$) still apply. So, with probability at least $(1 - \delta_1 - \delta_2)(1 - \delta_3) \geq (1 - \delta_1 - \delta_2 - \delta_3) = (1 - \delta)$, the error bound ϵ of Theorem 1 holds. Note however that all records contained

³ For our analysis, we assume that deletions are not adversarial, and specifically that they are pairwise-independent (to be explained in detail later in the paper).

in $\mathcal{D}$ are removed from the sketch. Therefore, the remaining records which may cause an estimation error are $\mathcal{I} \setminus \mathcal{D}$, and the effective stream length becomes $|\mathcal{I}| - |\mathcal{D}|$. $\square$

Complexity The space complexity of OmniSketch_B is $O(w \times d \times \beta \times K \times b \times |\mathcal{A}|)$. Inserting or deleting a sample comes with complexity $O(|\mathcal{A}| \times d \times \log(\beta K)/t)$, where t represents the size of the ingestion/deletion buffer (typically in the order of a few tens). Query execution takes $O(p \times d \times \beta K \times \log(\beta K))$ time.

Configuring the sketch Similar to $\mathbb{S}1$, the user sets the available memory M , and the desired values for ϵ and δ . The values of w and d are computed as follows: $w = 1 + \lceil e * ((\epsilon + 1)/\epsilon)^{1/d} \rceil = \Theta((1/\epsilon)^{1/d})$ and $d = \lceil \ln(3/\delta) \rceil$. The value of β is set following Lemma 4. The GitHub repository contains a module that computes the value of β , given the other parameters. In order to not exceed the memory quota M (in bits), K is set to the maximum value that satisfies $M \geq w * d * |\mathcal{A}| * (32 + \beta K * (\lceil \log(4(\beta K)^{2.5}/\delta) \rceil + 1))$. This value is computed efficiently again using the bisection method.

Adversarial deletes and support for general updates Our analysis assumed that deletions are pairwise independent – deletion of a record does not influence the probability of any other record to be deleted later in the stream. To illustrate the reasoning behind this assumption, consider the following example: OmniSketch_B is configured to allow up to 40% deletions (i.e., with $\alpha = 1/(1 - 40\%) \approx 1.67$), and is used to summarize a stream of 2 attributes $attr1, attr2$. The stream follows the bounded deletes model and satisfies this 40% deletions bound. Also, 25% of the records in the stream have $attr1 = v_1$. These records will be hashed to the same d cells of the sketch for $attr1$, and they will occupy around $25\% / (25\% + 75\%/w)$ of the samples in these cells. Indicatively, for $w = 10$, the records with $attr1 = v_1$ will occupy around 77% of the samples in the d cells corresponding to $attr1 = v_1$. If these insertions are followed by a targeted delete of all records with $attr1 = v_1$, these cells will lose around 77% of their samples, even though the number of deletes is still less than 40% of the stream. In such a case, the sketch may provide estimates that do not satisfy the accuracy guarantees.

There are multiple ways to handle this case. First, OmniSketch_B can easily detect and report this extreme event that the samples are depleted. The analysis can also be extended to anticipate for this case, if we know the distribution of deletions before observing the stream. However, we expect that an extension that works for the turnstile model with unbounded deletes (e.g., by replacing α -min-wise sampling with a deletion-resilient sampling algorithm like 2LHS [13]) would be a preferred approach. We are currently working on such an approach.

5 Experimental evaluation

The purpose of our experiment was (a) to compare the space complexity, efficiency, and accuracy of the proposed sketches to the state-of-the-art and to each other, and, (b) to examine how the sketches perform when summarizing streams of different characteristics, and in different configurations, both in the cash register and the bounded deletes model. We will first describe the used datasets, as well as different implementation concerns. We will then evaluate the sketches in two phases. Section 5.1 will focus on the sketches for the cash register model. Then, Section 5.2 will consider the sketches that can also handle deletes.

Datasets We used the CAIDA dataset [6], the SNMP dataset [17], the TPC-DS benchmark (table customer) [22], and a set of additional synthetic datasets to thoroughly evaluate particular properties of the compared algorithms. CAIDA (109 million records with 6 attributes) is a network flow monitoring dataset, whereas SNMP (8.2 Million records with 11 attributes) contains records collected from the wireless network of Dartmouth college during the fall of 2003. The TPC-DS benchmark comes with a dataset generator for evaluating performance of analytical queries. In our experiments, we used the *customer* table, generated with a scale factor 100,000. At this scale, the table contains 99 Million records. We then selected a subset of 5 attributes that are suited for use in predicate conditions with frequency queries. In addition to the above datasets, we generated synthetic datasets with different properties/distributions, to systematically evaluate how the properties of the input stream affect our algorithms. These datasets had up to 11 attributes, with the attribute values sampled from the uniform distribution and from Zipfian distributions parameterized with exponents $\alpha = \{1.0, 1.1, 1.3, 1.5\}$.⁴

Deletes Deletes were generated as follows. First, the dataset was randomly divided into two partitions: the noise partition p_n and the residue partition p_r . All insertions in the noise partitions were subsequently deleted, leaving only the records in the residue partition. The update operations (deletions and insertions) were then shuffled, with the only constraint being that each record had to be inserted before it was deleted. The deletes ratio controlled the proportion of noise insertions relative to total insertions, defined as $\frac{|p_n|}{|p_n| + |p_r|}$. Since each record in p_n was subsequently deleted, the total stream length was $2 * |p_n| + |p_r|$. Note that the total stream length increases significantly with the deletes ratio. For example, for a residue partition of 33 million records, a 50% deletes ratio results in a stream of 100 million updates

⁴ In Section 5 we use α to denote Zipf's exponent, to remain consistent with the literature. This α is unrelated to the one used in α -min-wise sampling of Section 4.

(inserts and deletes), while a 90% deletes ratio yields a stream that exceeds 637 million updates.

Hardware and implementation All experiments were executed on a Linux machine, equipped with 512 GB RAM and an Intel Xeon(R) CPU E5-2697 v2, clocked at 2.7GHz. The experiments were single-threaded, i.e., only one of the 48 cores was used, and the machine was otherwise idle. All sketches were implemented and executed in Java (OpenJDK version 19.0.2). For the Hydra baseline, we used the original code of the authors [21]. Since Hydra was originally constructed as a Spark plugin, to avoid unnecessary extra delays imposed by Spark, we extracted and used only the code that was necessary for centralized execution. Also, to ensure a fair comparison, we excluded from Hydra’s implementation any code and data structures associated with statistics beyond cardinality estimation, such as the L2 norm and entropy of sub-populations. Our changes improved Hydra’s performance and reduced its space complexity.

aSH [8] was also implemented in Java, single-threaded. We tested two versions of the algorithm – one with, and one without the buffer. As predicted in [8], the version with the buffer offered substantially higher throughput compared to the one without the buffer. Therefore, in the following we will only report results with the buffer. Additionally, for the TPC-DS stream that contained textual data of varying lengths, e.g., city name, we applied hashing to map these attributes to a fixed-length 32-bit integer. This also drastically improved the performance (both efficiency and effectiveness) of aSH, by enabling a fixed-size representation of the sampled attributes (4 bytes per attribute). This in turn: (a) eliminated the need for delimiter-based parsing of the samples, which introduces substantial overhead, and (b) allowed more records to be stored in the sample.

OmniSketch was also implemented in Java, single thread ed. Both attribute and min-wise hashing use 32-bit MurmurHash [15], with unique seeds per sketch row and repetition. We will be using the following naming to clearly distinguish between the different proposed OmniSketch variants: $S0_{\min}$ and $S0_{\cap}$ correspond to sketch $S0$ that uses the two different estimators presented in Section 3.1, whereas $S1$ and $S2$ correspond to the sketches and estimators presented in Section 3.2 and Section 4 respectively.

All algorithms were given a few seconds warm-up time (300 thousand updates), before we started to measure ingestion time. Since Java uses just-in-time bytecode compilation, this warm-up period was utilized to exclude the one-time cost of this compilation. The code of all methods, as well as detailed instructions on generating and processing the datasets and reproducing our results are available in github.⁵

⁵ The code and experiments related to the cash register model ($S0$ and $S1$) are available in <https://github.com/wiegerpunter/omnisketch>. The

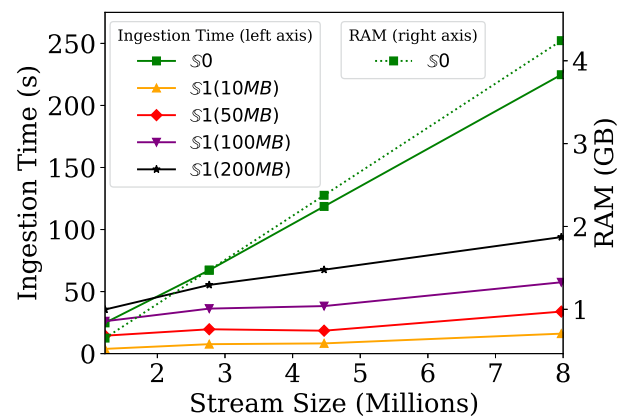


Fig. 4: Ingestion time and RAM for summarizing different stream lengths. The RAM sizes of different $S1$ -configurations are included in the legend

5.1 The cash register model

Our first set of experiments focused on evaluating our sketches that support insert-only streams (sketches $S0$ and $S1$). The purpose of the evaluation was: (a) to compare the space complexity, efficiency, and accuracy of $S0$ and $S1$ to each other and to the state-of-the-art, and, (b) to examine how our best performing sketch, $S1$, performs when summarizing streams of different characteristics, and in different configurations.

Queries The queries for each dataset were constructed as follows. At a pre-processing step, we went over a small sample of the dataset (roughly 0.05% of all records). For each sampled record, and for each possible number of predicates $p \in [2, |\mathcal{A}|]$, we constructed 10 queries by randomly choosing p attributes from the set of attributes $\mathcal{A}$, and their values in the sampled record as predicates. All queries were maintained in a set, effectively filtering out duplicates. This led to a total of 39,319 queries for the SNMP dataset on 11 distinct attributes, and 3,304 queries for CAIDA on 6 distinct attributes. All cash register results are averaged over one repetition of all queries per dataset.

5.1.1 Comparison of $S0$ and $S1$

Our first series of experiments was designed for comparing the two $S0$ estimators ($S0_{\min}$ and $S0_{\cap}$) with $S1$, in terms of (a) stream ingestion time, (b) memory requirements, (c) accuracy of the estimates, and, (d) query execution time. Accuracy was quantified by computing the mean absolute error, normalized by the stream size, i.e., $\sum_{q \in \mathcal{Q}} |\hat{f}(q) - f(q)| / (N * \dots)$

code related to the turnstile model ($S2$) is available in <https://github.com/wiegerpunter/omni-deletes.git>

Table 2: Query execution time in milliseconds, and estimation error

Algorithm	Query Time (ms)	Observed Error
$\mathbb{S}0_{\cap}$	20.886	4×10^{-6}
$\mathbb{S}0_{\min}$	29.867	0.0038
$\mathbb{S}1$ (10 MB)	0.227	0.0004
$\mathbb{S}1$ (50 MB)	1.212	0.0003
$\mathbb{S}1$ (100 MB)	2.452	0.0003
$\mathbb{S}1$ (200 MB)	6.620	0.0003

$|\mathbb{Q}|$), where N denotes the stream size and $\mathbb{Q}$ denotes the set of executed queries. For this set of experiments, all sketches were configured with $\epsilon = 0.1$ and $\delta = 0.1$. We tested four different configurations of $\mathbb{S}1$, with 10, 50, 100, and 200 MB of RAM.

Figure 4 shows the required time and memory for summarizing subsets of the SNMP stream (all 11 attributes), with the compared sketches. Recall that $\mathbb{S}0_{\min}$ and $\mathbb{S}0_{\cap}$ use the same sketch, and are therefore presented together as $\mathbb{S}0$ in this figure. $\mathbb{S}1$ memory requirements are not included in the figure as separate series, since these are fixed for each configuration. As expected, we see that the required time for building all sketches (left axis) grows linearly with the stream size. Sketch $\mathbb{S}1$ is however significantly more efficient, requiring 4 to 5 times less time compared to $\mathbb{S}0$. Also, the throughput of $\mathbb{S}1$ decreases when the sketch is allowed to use more memory. This is again expected, since a higher memory quota for $\mathbb{S}1$ translates to a higher value of B (more samples per cell), which leads to slower maintenance of the red-black tree. Still, even for $\mathbb{S}1$ with 200 MB RAM and 11 attributes, throughput exceeds 89 thousand updates per seconds. Also notice that $\mathbb{S}1$ ingestion time forms a subtle, yet visible, elbow (e.g., for $\mathbb{S}1$ with 200 MB, this happens at around 2.7 Million updates). This elbow signifies that the K min-wise samples at most cells in the sketch reached to an almost *stable* state, and the probability that any new update needs to be added to the samples is small. At this point, most updates take only $O(|\mathcal{A}| \times d)$ time, as opposed to $O(\log(B) \times |\mathcal{A}| \times d)$. Finally, we observe that the memory requirements of $\mathbb{S}0$ (Fig. 4, right axis) grow linearly with the stream size, since all records are kept in the sketch.

Table 2 summarizes the average error and query execution time for all considered configurations. We observe that both $\mathbb{S}0$ variants take significantly more time for query execution compared to the $\mathbb{S}1$ variants. The difference in performance is somewhere between 1 and 2 orders of magnitude, depending on the memory quota of $\mathbb{S}1$. The reason for this stark difference is because the $\mathbb{S}0$ variants need to iterate over very large sets for computing the intersection, whereas $\mathbb{S}1$ iterates

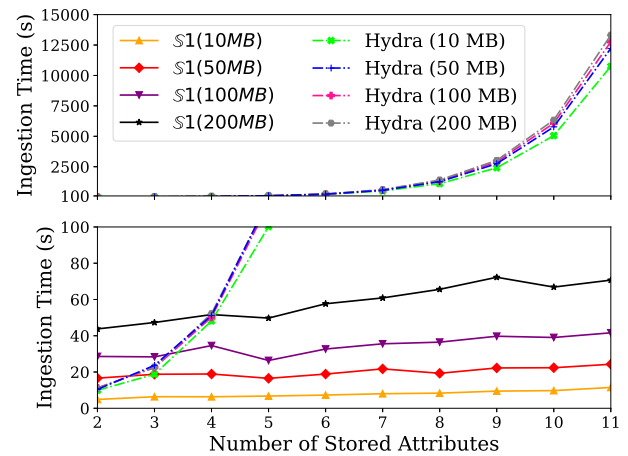


Fig. 5: Ingestion Time for Hydra and $\mathbb{S}1$, across vertical partitions (varying number of attributes) of the SNMP stream. For clarity, the plot is split into two sub-plots, with different Y-axis scales

over samples of size B , with B in the order of a few tens of thousands. We also see that $\mathbb{S}0_{\cap}$ estimator is notably faster than $\mathbb{S}0_{\min}$, even though both operate on the same sketch structure. This is attributed to the way the query execution is implemented in the two estimators. In $\mathbb{S}0_{\cap}$, similar to $\mathbb{S}1$, we start from the first cell, and keep reducing the candidate records until we intersect all $p \times d$ cells (see the relevant discussion in the implementation of $\mathbb{S}1$, Section 3.2). This quickly reduces the candidate records. On the other hand, the implementation of the $\mathbb{S}0_{\min}$ estimator performs the same process *per row*, and takes the minimum value of all rows. Therefore, $\mathbb{S}0_{\min}$ repeats each check multiple times, for each record that satisfies the query.

In terms of approximation error, $\mathbb{S}0_{\cap}$ outperforms all others, with a negligible average error. This is to be expected, since $\mathbb{S}0_{\cap}$ does not suffer from underestimations and false negatives, and its probability of including false positives in the estimate is very small: $\epsilon^{p'd}$, where p' denotes the number of query attributes not satisfied by the record. We also see that $\mathbb{S}1$, which relies on the intersection, has around an order of magnitude smaller error than $\mathbb{S}0_{\min}$, even though it requires one to two order of magnitudes less memory for representing the same dataset.

Summary The comparison between $\mathbb{S}0_{\min}$, $\mathbb{S}0_{\cap}$ and $\mathbb{S}1$ revealed that $\mathbb{S}0_{\cap}$ substantially outperforms the other two variants in terms of accuracy, but with linear space requirements. Therefore, $\mathbb{S}0_{\cap}$ is only useful as an alternative to full indexing, where there is sufficient memory for storing the whole stream. The size of $\mathbb{S}0_{\min}$ also grows linearly with the data, and the estimator also performs worse compared to $\mathbb{S}0_{\cap}$. On the other hand, $\mathbb{S}1$ offers a favorable trade-off between

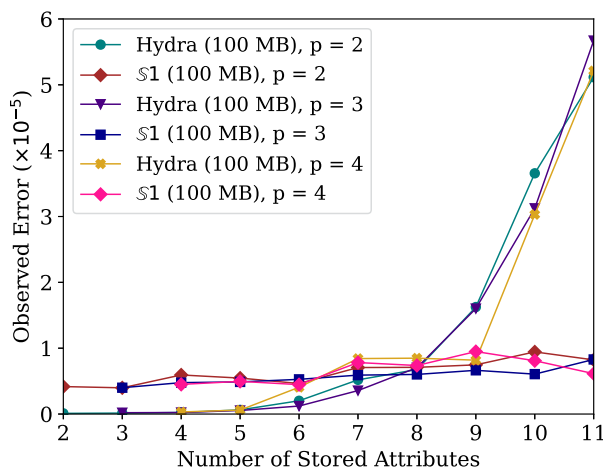


Fig. 6 : Estimation error of Hydra and S1, on different vertical partitions of SNMP

Table 3 : Query Execution Time in milliseconds, for queries with different numbers of predicates p

# query pred.	2	4	6	8
S1 (10 MB)	0.18	0.13	0.12	0.09
S1 (50 MB)	0.98	0.79	0.68	0.44
S1 (100 MB)	2.02	1.61	1.25	1.13
S1 (200 MB)	6.38	4.33	2.85	2.64
Hydra (10 MB)	0.04	0.04	0.02	0.02
Hydra (50 MB)	0.05	0.04	0.03	0.00
Hydra (100 MB)	0.05	0.04	0.02	0.02
Hydra (200 MB)	0.05	0.04	0.02	0.00

accuracy and space complexity/efficiency, and allows the user to fine-tune the memory quota in order to fully utilize the available RAM. Therefore, for the remaining experiments and comparison with the state-of-the-art we will only consider S1.

5.1.2 Comparison with the state-of-the-art

Our second series of experiments focused on comparing S1, our best-performing sketch, with Hydra. The two sketches were allowed the same memory, and were compared on their (a) ingestion time, (b) query execution performance, and, (c) accuracy.

Ingestion time High throughput is the key requirement for stream processing and summarization. Hence, our initial experiment aimed to assess the time required by S1 and Hydra to summarize the entire stream, and to examine how this time was affected by the number of attributes in the stream. We generated streams with varying number of attributes (from

2 to 11) by taking distinct vertical partitions of SNMP (i.e., different subsets of the available attributes).

Figure 5 shows the ingestion time of Hydra and S1, for different memory quotas. Our first observation is that both algorithms have comparable performance when storing up to 4 attributes. However, the computational complexity of Hydra grows exponentially with the number of stream attributes. This happens because each record in Hydra leads to $2^{|\mathcal{A}|} - 1$ sketch updates, where $\mathcal{A}$ denotes the set of stream's attributes. Therefore, storing all 11 attributes of SNMP in Hydra leads to $O(2^{11})$ sub-populations/updates per record, and takes around 14 thousand seconds for the full dataset (more than 1 msec per record). It is important to note here that storing all sub-populations is required in Hydra, not only for supporting queries with exactly 11 predicates, but also for supporting queries that contain subsets of these predicates whenever the exact combinations of the query predicates is not known before observing the stream.

Interestingly, addition of more attributes has a negligible impact on the ingestion time of S1. In fact, there exist cases where adding an attribute does not increase, or even slightly reduces ingestion time (e.g., when adding the fifth attribute for S1, 200 MB). At first sight, this result may appear counter-intuitive, since adding more attributes means maintaining more attribute sketches, and storing more samples in these sketches. This behavior is attributed to the fact that the most time-consuming step of adding a record at S1 is the one of maintaining the K min-wise samples, which becomes faster when the sample size B is reduced. By adding more attributes in the sketch without increasing its memory quota, we effectively reduce the sample size B for all of the sketch's attributes. This leads to: (a) reduction of the probability that the red-black tree needs to be updated, and, (b) faster updates of the red-black tree, whenever these are required.

Accuracy Figure 6 plots the observed error on the two sketches, for queries with $p = 2, 3$, and 4 predicates. We see that even though the number of query predicates does not significantly influence the accuracy of either method, the number of stored attributes does influence the accuracy of Hydra. Beyond 8 attributes, Hydra's accuracy is reduced, whereas the accuracy of S1 is not affected. This behavior of Hydra is again attributed to the number of inserts that Hydra eventually performs per record: the information summarized by Hydra increases exponentially to the number of attributes, leading to more collisions and to a drastic increase of the observed error.

Query execution time For the final comparison, we used both Hydra and S1 to summarize all 11 attributes of the whole SNMP stream (8.3 Million updates). Table 3 summarizes the query execution time for queries with up to 8 predicates. Both sketches are very efficient, requiring less than 10 msec for executing a query, even in the configuration

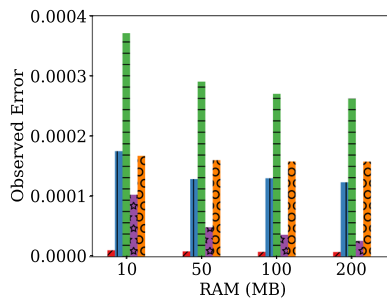


Fig. 7 : Observed error of S1 for different streams. The error at the Uniform stream was close to 0 and is thus omitted from the figure. Legend identical to Fig. 8

with a 200 MB RAM quota. We also note that S1 is slower than Hydra. This is expected, since S1 needs to compute the intersection of $p \times d$ samples of size B , whereas Hydra only needs to compute the minimum of an approximately equal number of counters. For the same reason, the query time for S1 also grows slightly with the available memory, whereas Hydra performance stays unaffected. Also notice that S1's efficiency increases with the number of predicates, aligning with our earlier observation (Section 5.1.1). This improvement is attributed to the algorithm's way of computing the sets intersection, which becomes more efficient as the number of attributes/sets increases (see Section 3.2).

Summary Both Hydra and S1 require less than 10 msec to execute queries with up to 8 predicates. However, S1 substantially outperforms Hydra in efficiency when summarizing streams with many attributes, with a difference in throughput that may exceed two orders of magnitude. Importantly, Hydra's throughput rate decreases exponentially with the number of stream attributes. The same trend of exponential decrease with the number of stream attributes is observed on Hydra's estimation accuracy, even for queries that contain very few attributes. Therefore, Hydra is not a viable option for summarizing streams that contain many attributes.

5.1.3 Evaluation of S1 with different streams

Our next set of experiments investigates the effect of the stream properties (distribution of the attribute values and number of records in the stream) to S1's efficiency and accuracy.

Figure 8 shows the required time for summarizing 5 Million records from the following streams:⁶

- **Zipf**, $\alpha \in [1, 1.3, 1.5]$: Three different streams. Each of these streams contains 5 attributes (integer values). We

⁶ Details for the generation of all streams are included in the project's github

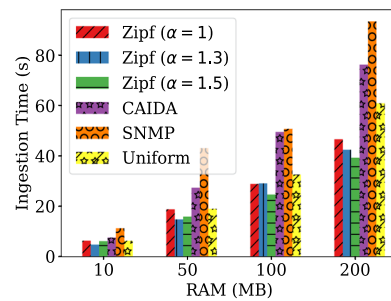


Fig. 8 : Ingestion time of S1 for different streams

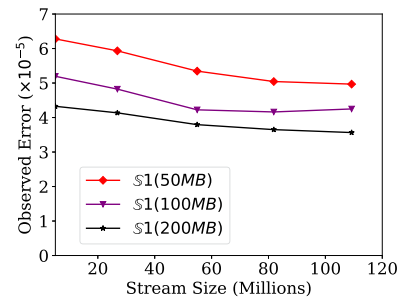


Fig. 9 : Estimation error of S1 for streams of different lengths

generate the records by drawing random numbers from Zipfian distributions with exponent $\alpha \in [1, 1.3, 1.5]$.

- **Uniform**: The stream contains 5 integer attributes. The values for all attributes are drawn by a uniform distribution.
- **CAIDA and SNMP**: A vertical partitioning of the CAIDA and SNMP streams, containing 5 attributes.

We see that S1 requires less than 100 seconds to summarize each stream, even for the 200 MB quota. Also, the ingestion time for the Zipf streams is notably smaller compared to the uniform stream, and is further reduced as the Zipf's α value increases. This behavior can be traced back to the effort required for maintaining the K min-wise samples, which aligns with our earlier observation of the subtle elbow in Figure 4. As the Zipf exponent value rises in this experiment, the distribution of values becomes more skewed, causing more records to hash into the same cells at the corresponding attribute sketches. Consequently, the K min-wise samples of these cells swiftly reach an almost stable state. This pattern is also observed in the CAIDA and SNMP streams, where certain attributes also follow a Zipfian distribution, resulting in faster ingestion compared to the uniform stream. Still, even for the extreme case that the values for all stream attributes follow a uniform distribution, S1 throughput surpasses 150k updates per second.

The observed estimation error on these datasets is shown in Figure 7. The presented results correspond to the average error over 450 queries containing three predicates. We see that the observed error remains very low in all cases. For the case of Uniform, the average error is close to 0, whereas the highest error is observed with Zipf, $\alpha = 1.5$. The reason for this is because, as the Zipf exponent grows, the samples of the popular cells in the sketch (the cells responsible for storing the values with frequencies at the head of the Zipf distribution) end up containing almost exclusively records of these popular values. As a result, not sufficient samples remain for low-frequency predicates that happen to fall in the same popular cells, leading to higher estimation errors. This limitation is common across all small-memory sketches that provide error relevant to the stream size, e.g., the Count-min sketch [11].

Our final experiment for the cash register model aimed to explore the impact of the stream size to the performance of $\mathbb{S}1$. Since we already demonstrated that the throughput of $\mathbb{S}1$ is unaffected with the stream size (Figure 4), this experiment focused solely on the impact of the stream size to accuracy. Precisely, we used $\mathbb{S}1$ to summarize CAIDA, which was the largest real dataset with 109 Million records. At regular intervals, we paused the stream ingestion, executed a fixed set of 3304 queries on the sketch, and computed the estimation error per query. The depicted results, shown in Figure 9, correspond to the mean observed error per interval. As predicted by the analysis, the observed error in OmniSketch stays stable with the stream size.

Summary $\mathbb{S}1$ is efficient in all cases, with a throughput that exceeds 150k updates per second. In terms of efficiency, the most difficult distribution is the uniform distribution, because it takes longer for the samples in the sketch cells to reach to a stable state. In terms of accuracy, Zipf distributions with extremely high values are more difficult for $\mathbb{S}1$. This is a limitation shared across all small-memory sketches. Furthermore, increasing the stream size does not have a significant influence on the observed error.

5.2 The bounded deletes model

Our second round of experiments focused on (a) comparing $\mathbb{S}2$ to $\mathbb{S}1$, the best performing sketch in the previous round of evaluations, (b) evaluating $\mathbb{S}2$ against aSH, the state-of-the-art competitor for summarizing streams under the turnstile model, and, (c) assessing the performance of $\mathbb{S}2$ on streams and queries of different characteristics.

Queries and algorithms' configuration Queries were methodically constructed, to comprehensively cover a broad spectrum of query characteristics (different numbers of query predicates and results). Queries were created by: (a) selecting random records from the dataset, (b) generating multiple

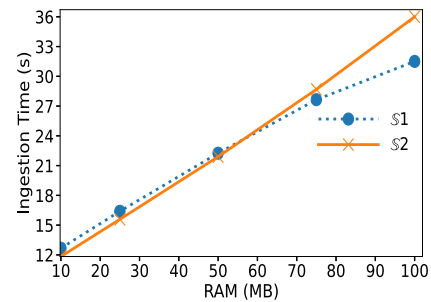


Fig. 10 : Ingestion time of $\mathbb{S}1$ and $\mathbb{S}2$ on the CAIDA stream, for different sketch sizes

query instances per selected record, by masking a varying number of attributes to simulate different predicate counts. This masking process ensured coverage across queries with different levels of selectivity, and with different number of predicates.

For these experiments, $\mathbb{S}2$ was configured with 512 KB allocated to insertion buffers and 512 KB to deletion buffers *across all cells*, i.e., a total of 1 MB allocated for buffering in the whole sketch. For aSH, a single 1 MB buffer was used, which was sufficient for storing more than 87K records of 3 attributes.⁷ The reported RAM for both algorithms includes the 1 MB allocated to the buffers. All bounded-deletes results are averaged over five repetitions.

5.2.1 Comparison of $\mathbb{S}1$ and $\mathbb{S}2$

Recall that $\mathbb{S}2$ can also be used for insert-only streams. When initialized with $\beta = 1$, $\mathbb{S}2$ replicates the behavior of $\mathbb{S}1$, using the same estimators and providing identical error guarantees and estimates, assuming the sample size remains the same in the two sketches. However, $\mathbb{S}2$ introduces several algorithmic optimizations (see Section 4.1.1) that alter how samples are organized and accessed. These changes affect both the number of samples that can be stored within the memory quota – hence also the accuracy of the estimates – and the ingestion and query throughput. In this set of experiments, we compare the performance and accuracy of the two sketches to assess the impact of these optimizations

Figures 10-12 show the ingestion time, query time, and observed error of the two sketches across various sketch sizes, on the CAIDA stream without deletions. Our first observation is that both sketches exhibit comparable ingestion throughput, even though $\mathbb{S}2$ maintains a sorted array of primitives rather than a red-black tree. These results are consistent for different RAM sizes. At first glance, this result is surprising, as maintaining a sorted array incurs non-negligible memory

⁷ Larger buffer sizes would render continuous query execution impractical under aSH since the buffer needs to be flushed before each query.

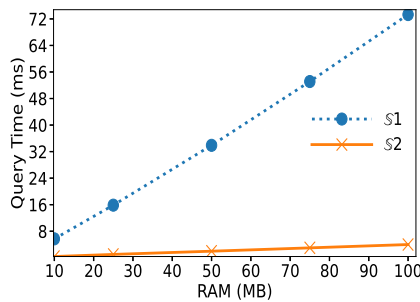


Fig. 11 : Query execution time of S1 and S2 on the CAIDA stream, for different sketch sizes

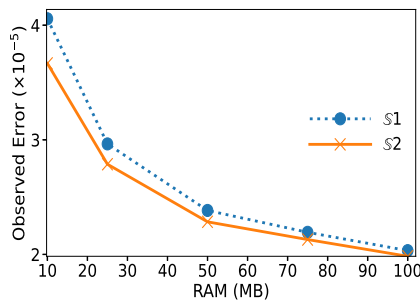


Fig. 12 : Observed error of S1 and S2 on the CAIDA stream, for different sketch sizes

copy costs when inserting new samples. This performance can be attributed to the two buffers used in S2 for insertion and deletion, which group updates into larger batches, significantly reducing the sizes of these of memory copies Fig 11.

In terms of query time, S2 outperforms S1 by an order of magnitude. This improvement is again due to the organization of samples in S2, which enhances memory locality and enables faster computation of the intersections of the samples: in S2, all records within a sample are stored contiguously in memory, whereas S1's red-black tree organization of samples requires pointer traversal to iterate over the samples.

Finally, we see that S2 offers noticeably improved accuracy compared to S1 (Figure 12), when both are allocated the same memory. This is again attributed to the improved representation of the samples as an array, which avoids storing pointers between the samples, thereby allowing more records to be included in the samples.

Summary We see that S2 is comparable to or outperforms S1 across all metrics when summarizing streams that follow the cash register model. These performance gains are attributed to the contiguous organization of samples in a sorted array, enabled by the newly introduced insertion and deletion buffers (Section 4.1.1). Based on these results, we exclude S1 and all other sketches it outperformed in Sec-

tion 5.1 from the remaining experiments, and focus on S2 and aSH.

5.3 Comparison of S2 with aSH

We now compare S2 with aSH, the state-of-the-art algorithm for summarizing streams with deletions. In the following experiments, all streams include 50% deletions, i.e., half of all inserted elements were later deleted, as described in Section 5.

Ingestion Time Figures 13-15 show the ingestion time of both algorithms across different memory quotas, using the CAIDA, TPC-DS, and a synthetic dataset. The synthetic dataset is generated with a 50% deletes ratio, and contains a total of 20.1 million updates – approximately 13.4 million insertions out of which half of them were later deleted. Attribute values in the synthetic datasets follow the Zipf distribution with exponent 1.1. CAIDA and TPC-DS also have a 50% deletes ratio. SNMP was excluded from the experiments with deletions, as it was not large enough to allow for a meaningful summarization after 50% of the records were deleted. For each dataset we examined three vertical partitions: for CAIDA and TPC-DS, we constructed partitions of 3, 4, and 5 attributes, whereas for the synthetic dataset, which contains 11 attributes, we used partitions of 3, 6, and 11 attributes. To allow for a meaningful comparison between vertical partitions, all queries were created using a maximum of three predicates – the ones included in the smallest partition of each dataset Fig 14.

We observe that the ingestion time of aSH is at least an order of magnitude higher than that of S2. Ingestion time increases linearly with the memory quota for both algorithms. This behavior is expected, as a larger memory budget allows the two algorithms to maintain larger samples, which increases maintenance overhead. In the case of aSH, the entire sample must be scanned whenever records are evicted, whereas for S2, maintaining a larger sorted array leads to more expensive memory copies during insertions. The impact is more pronounced for aSH for two key reasons: (a) aSH maintains a single large sample, whereas S2 maintains $w \times d \times |\mathcal{A}|$ smaller samples that can be updated more efficiently, and, (b) aSH must scan its entire sample twice during each eviction cycle, while S2 leverages binary search and the sorted array to reduce this cost.

Query execution time Figures 16-18 plot the query execution time for the same input streams. We see that S2 is several orders of magnitude faster than aSH. This is expected, as aSH must scan its entire sample for each query, whereas S2 intersects only a few smaller sorted samples using leapfrog join. Also notice that aSH query time increases when fewer attributes are indexed, as this leads to larger samples with more records that need to be traversed for answering the

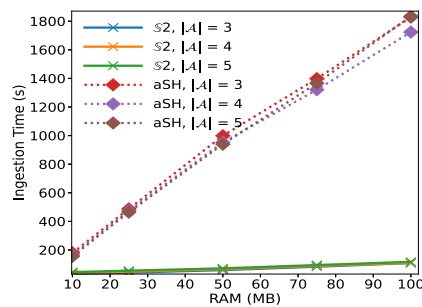


Fig. 13 : Ingestion time of aSH and S2, for ingesting different vertical partitions (varying the number of attributes) of the CAIDA stream

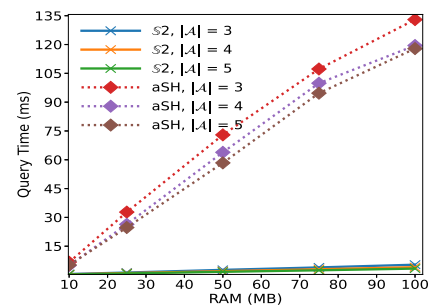


Fig. 16 : Query execution time of aSH and S2, on different vertical partitions (varying the number of attributes) of the CAIDA stream

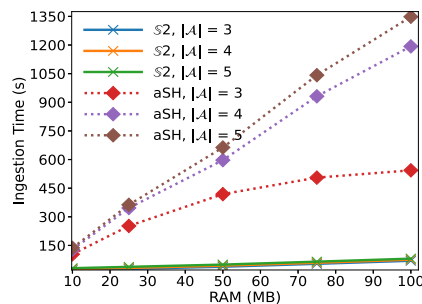


Fig. 14 : Ingestion time of aSH and S2, for ingesting different vertical partitions (varying the number of attributes) of the TPC-DS stream

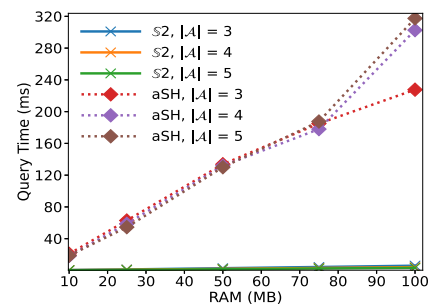


Fig. 17 : Query execution time of aSH and S2, on different vertical partitions (varying the number of attributes) of the TPC-DS stream

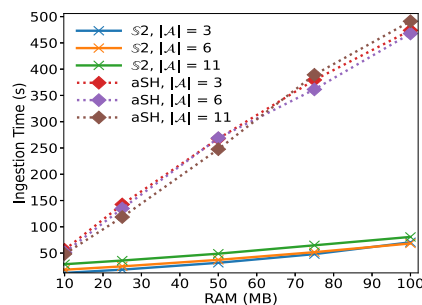


Fig. 15 : Ingestion time of aSH and S2, for ingesting different vertical partitions (varying the number of attributes) of the synthetic stream with Zipf ($\alpha = 1.1$)

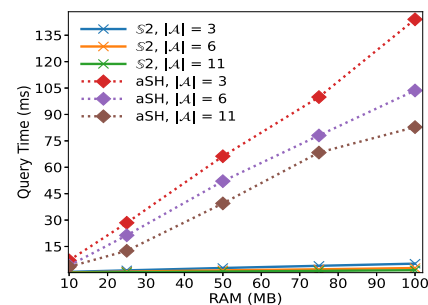


Fig. 18 : Query execution time of aSH and S2, on different vertical partitions (varying the number of attributes) of the synthetic stream with Zipf ($\alpha = 1.1$)

query. Nevertheless, even at the lowest memory quota, aSH remains significantly slower than S2 Fig. 17.

Accuracy Figures 19–21 show the average observed error for different memory quotas and vertical partitions for the three datasets. Our first observation is that the accuracy of both synopses improves with increased memory. The observed error is negligible across all datasets, though the synthetic dataset is noticeably more difficult for both algorithms. aSH exhibits slightly better accuracy on CAIDA and

TPC-DS, whereas S2 achieves comparable or slightly better accuracy on the synthetic dataset. Overall, both algorithms offer comparable accuracy across configurations, with differences typically on the order of 10^{-5} .

Summary Both aSH and S2 offer comparable accuracy with observed error differences on the order of 10^{-5} . However, S2 significantly outperforms aSH in ingestion throughput (by one to two orders of magnitude) and in query execution time (by at least two orders of magnitude). There-

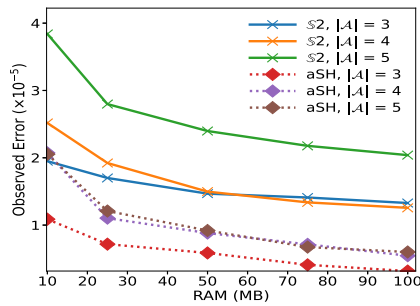


Fig. 19 : Estimation error of aSH and $\mathbb{S}2$, on different vertical partitions (varying the number of attributes) of the CAIDA stream

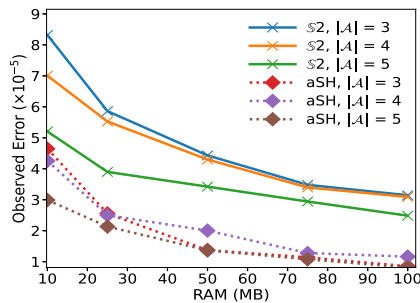


Fig. 20 : Estimation error of aSH and $\mathbb{S}2$, on different vertical partitions (varying the number of attributes) of the TPC-DS stream

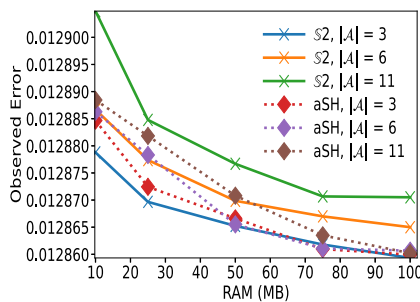


Fig. 21 : Estimation error of aSH and $\mathbb{S}2$, on different vertical partitions (varying the number of attributes) of the synthetic stream with Zipf ($\alpha = 1.1$)

fore, even though both algorithms are sufficiently accurate, the increased efficiency of $\mathbb{S}2$ can be instrumental in high-throughput or latency-sensitive applications, where ingestion and querying speed is critical.

5.4 Evaluation of $\mathbb{S}2$ in different configurations

Our final round of experiments investigates the impact of stream properties on the efficiency and accuracy of $\mathbb{S}2$. To

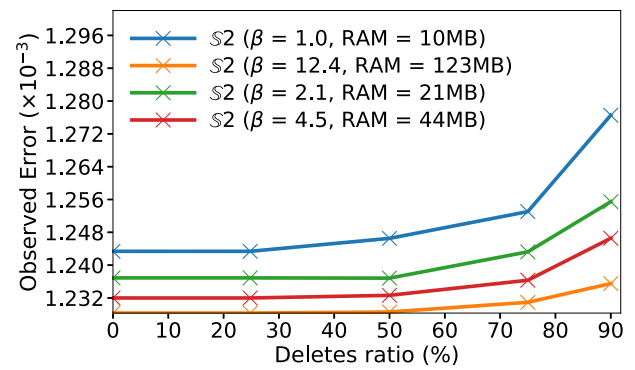


Fig. 22 : Estimation error of $\mathbb{S}2$ for varying deletes ratios on a synthetic data stream with Zipf ($\alpha = 1.1$) and 11 attributes

this end, we consider (a) streams with varying deletes ratios, (b) streams with different attribute value distributions, and, (c) queries with varying number of predicates.

For the first experiment, we generated streams with varying deletes ratios, from 0% to 90%. The total length of the stream therefore varied from 33 million updates (2^{25}) to 637 million updates. The residue partition (size 33 million) and the queries were in all cases identical after the completion of the stream, to allow for a meaningful comparison. We evaluated four configurations of $\mathbb{S}2$ with different values of β , ranging from 1.0 (no provision for deletions) to 12.4 (supporting up to 90% deletions). Figures 20 and 22 shows the observed error for these configurations. Our first observation is that the error remains low even with 90% deletes – typically on the order of 1×10^{-3} . We also see that the choice of β affects performance; with $\beta = 1$ (no provision for deletions), the error increases noticeably after just 25% deletions, and grows sharply at 75% deletions. In contrast, with $\beta = 12.4$ (support for 90% deletions) the error increase only becomes noticeable after 50% deletions, and remains more gradual overall. Higher β values also improve accuracy even on streams with no deletions. This is due to the way the queryable sample is decided in α -min-wise sampling (see Section 4), which allows the sketch to use K' samples at each cell for answering the query, where K' can be significantly larger than K .

Figure 23 shows the ingestion time for $\mathbb{S}2$, when summarizing CAIDA, TPC-DS, and the synthetic dataset ($\alpha \in \{1.1, 1.3, 1.5\}$), all configured with 50% deletions, and a total length of 20 million updates to be able to have a meaningful comparison. For these experiments, the synthetic dataset contained 11 attributes, to allow for queries with up to 11 predicates. We see that increasing the Zipf exponent slightly improves ingestion throughput. This observation aligns with our previous findings for $\mathbb{S}1$ (Section 5.1.3): datasets with higher Zipf exponents produce more skewed distribution of

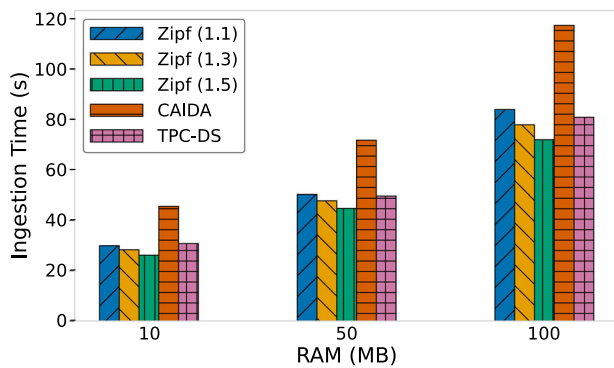


Fig. 23 : Ingestion time of S2 for different data streams. All streams have 50% deletes and index the maximum number of attributes in the stream

Table 4 : Estimation error of S2 for different data streams

RAM (MB)	10	50	100
Zipf (1.1)	1.25E-3	1.23E-3	1.23E-3
Zipf (1.3)	2.55E-3	2.50E-3	2.49E-3
Zipf (1.5)	5.48E-3	5.37E-3	5.35E-3
CAIDA	3.84E-5	2.40E-5	2.04E-5
TPC-DS	5.21E-5	3.42E-5	2.49E-5

the records to the sketch's cells, which leads to the corresponding samples reaching a stable state faster. Compared to the insert-only streams and S1, this difference is less pronounced in S2, as deletions in the stream still introduce additional modifications to the sketch. A similar pattern is observed for TPC-DS, while CAIDA's less skewed distribution results in slightly lower ingestion throughput.

Table 4 reports the observed query error for these streams. For each dataset, the number of query predicates was selected uniformly at random between 1 and the total number of available attributes. The results align with our previous findings (Figures 8 and 21), showing that: (a) the synthetic dataset is the most challenging, with difficulty increasing as the Zipf exponent grows, (b) increasing the memory quota consistently reduces the error, and, (c) even under highly-skewed Zipf distributions, the observed error remains very low.

Finally, Table 5 summarizes the observed error and query execution time for queries involving varying numbers of predicates. The results correspond to the synthetic dataset with 50% deletes ratio, generated with Zipf exponent 1.1 and a total of 20 million updates. We observe that accuracy improves significantly as the number of predicates increases. This is attributed to a reduction in the number of false positives, due to the additional intersections. Query execution time remains stable – around 0.15 msec for the 10 MB sketch and around 2 msec for the 100 MB sketch, indicating that

Table 5 : Average observed error and query time, varying the number of predicates per query

RAM (MB)	10	100	10	100
#Predicates	Observed Error		Query Time (ms)	
1	1.99E-2	1.97E-2	0.24	1.75
3	3.91E-4	3.85E-4	0.13	1.60
5	2.60E-7	3.50E-7	0.13	1.63
7	1.02E-6	2.20E-7	0.13	1.59
9	1.50E-7	2.10E-7	0.14	1.72
11	1.50E-7	1.50E-7	0.2	1.90

increasing the number of predicates has minimal impact on computational cost.

Summary S2 remains both efficient and accurate across all datasets, even with up to 90% deletions. As with S1 and other small-memory synopses, highly skewed attribute values distributions (e.g., Zipf with a skew of 1.5) pose the greatest challenge for the sketch in terms of accuracy. Nonetheless, S2 continues to estimate the answers with very small errors, while maintaining support for high-throughput ingestion and fast query execution. Increasing the number of predicates in a query further reduces the probability of a false positive to survive, leading to improved accuracy, with minimal impact on query latency.

6 Conclusions

We introduce OmniSketch, a compact and efficient sketch for summarizing the distributions of complex, high-dimensional data streams with deletions. OmniSketch combines a small (user-defined) memory footprint with fast update and query times, and offers theory-backed accuracy guarantees. The final sketch builds on α -min-wise sampling, a new sampling method introduced in this work. α -min-wise sampling is the first min-wise hashing algorithm that supports deletions, comes with theory-backed accuracy guarantees, and is of independent interest beyond its use in OmniSketch – for example, as a drop-in replacement in systems that use other min-wise hashing algorithms, enabling them to handle streams with deletions. We further introduce a set of algorithmic optimizations that significantly boost performance, enabling throughput at least an order of magnitude higher than existing approaches. Extensive experiments across diverse datasets confirm that OmniSketch outperforms the state-of-the-art methods in both ingestion and query efficiency, while maintaining comparable accuracy.

Acknowledgements This work was partially funded by the European Commission under the STELAR (HORIZON-EUROPE - Grant

No. 101070122) and DataTools4Heart (HORIZON-HLTH - Grant No. 101057849) projects.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.* **58**(1), 137–147 (1999)
- Becchetti, L., Clementi, A., Gualá, L., Sciarria, L.P., Straziota, A., Stromieri, M.: Approximate 2-hop neighborhoods on incremental graphs: An efficient lazy approach. *Proc. VLDB Endow.* **18**(11), 3937–3950 (2025). <https://doi.org/10.14778/3749646.3749665>
- Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **13**(7), 422–426 (1970). <https://doi.org/10.1145/362686.362692>
- Broder, A.Z.: On the resemblance and containment of documents. In: *Proc. of Compression and Complexity of Sequences*, pp. 21–29. IEEE (1997). <https://doi.org/10.1109/SEQUEN.1997.666900>
- Bury, M., Schwegelshohn, C., Sorella, M.: Similarity search for dynamic data streams. *IEEE TKDE* **32**(11), 2241–2253 (2020). <https://doi.org/10.1109/TKDE.2019.2916858>
- CAIDA: The CAIDA UCSD Anonymized Internet Traces Dataset (accessed October 2023) (2019). https://www.caida.org/catalog/datasets/passive_dataset/
- Clementi, A., Gualá, L., Pepè Sciarria, L., Straziota, A.: Maintaining k-minhash signatures over fully-dynamic data streams with recovery. In: *Proc. of WSDM*, p. 79–87 (2025). <https://doi.org/10.1145/3701551.3703491>
- Cohen, E., Cormode, G., Duffield, N.G.: Don't let the negatives bring you down: sampling from streams of signed updates. In: *ACM SIGMETRICS/PERFORMANCE*, pp. 343–354 (2012). <https://doi.org/10.1145/2254756.2254798>
- Cohen, E., Kaplan, H.: Tighter estimation using bottom k sketches. *Proc. VLDB Endow.* **1**(1), 213–224 (2008). <https://doi.org/10.14778/1453856.1453884>
- Cormode, G., Garofalakis, M.N., Haas, P.J., Jermaine, C.M.: Synopses for massive data: Samples, Histograms, Wavelets. *Sketches. Found. Trends Databases* **4**(1–3), 1–294 (2012). <https://doi.org/10.1561/1900000004>
- Cormode, G., Muthukrishnan, S.: An improved data stream summary: the Count-Min sketch and its applications. *J. Algorithms* **55**(1), 58–75 (2005). <https://doi.org/10.1016/J.JALGOR.2003.12.001>
- Flajolet, P., Éric Fusy, Gandouet, O., Meunier, F.: HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. *DMTCS AH*, 10 (2007). <https://doi.org/10.46298/dmtcs.3545>
- Ganguly, S., Garofalakis, M.N., Rastogi, R.: Tracking set-expression cardinalities over continuous update streams. *VLDB J.* **13**(4), 354–369 (2004). <https://doi.org/10.1007/S00778-004-0135-3>
- Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.: How to summarize the universe: Dynamic maintenance of quantiles. In: *Proc. VDLB*, pp. 454–465 (2002). <https://doi.org/10.1016/B978-155860869-6/50047-0>
- Google: Guava: Google core libraries for java. <https://github.com/google/guava> (2024). Version used: 33.3.1
- Jayaram, R., Woodruff, D.P.: Data streams with bounded deletions. In: *Proc. PODS*, pp. 341–354. ACM (2018). <https://doi.org/10.1145/3196959.3196986>
- Kotz, D., Henderson, T., Abyzov, I., Yeo, J.: CRAWDAD Dartmouth/campus (v. 2009-09-09) (2022)
- Li, P., König, A.C.: b-Bit minwise hashing. In: *Proc. WWW*, pp. 671–680. ACM (2010). <https://doi.org/10.1145/1772690.1772759>
- Li, P., Owen, A.B., Zhang, C.: One permutation hashing for efficient search and learning. *CoRR abs/1208.1259* (2012)
- Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., Braverman, V.: One sketch to rule them all: Rethinking network flow monitoring with univmon. In: *Proc. SIGCOMM*, pp. 101–114 (2016). <https://doi.org/10.1145/2934872.2934906>
- Manousis, A., Cheng, Z., Basat, R.B., Liu, Z., Sekar, V.: Enabling efficient and general subpopulation analytics in multidimensional data streams. *Proc. VLDB Endow.* **15**(11), 3249–3262 (2022). <https://doi.org/10.14778/3551793.3551867>
- Nambiar, R.O., Poess, M.: The making of TPC-DS. In: *Proc. VLDB Endow.*, VLDB '06, p. 1049–1058. VLDB Endowment (2006)
- Pagh, R., Stöckel, M., Woodruff, D.P.: Is min-wise hashing optimal for summarizing set intersection? In: *Proc. PODS*, pp. 109–120. ACM (2014). <https://doi.org/10.1145/2594538.2594554>
- Punter, W.R., Papapetrou, O., Garofalakis, M.N.: OmniSketch: efficient multi-dimensional high-velocity stream analytics with arbitrary predicates. *Proc. VLDB Endow.* **17**(3), 319–331 (2023)
- Rice, J.A.: *Mathematical statistics and data analysis*, vol. 371. Thomson/Brooks/Cole Belmont, CA (2007)
- Veldhuizen, T.L.: Triejoin: A simple, worst-case optimal join algorithm. In: *ICDT*, pp. 96–106. OpenProceedings.org (2014)
- Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M.e.a.: Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature Methods* **17**(3), 261–272 (2020). <https://doi.org/10.1038/s41592-019-0686-2>
- Zhang, F., Wang, S.: Effective indexing for dynamic structural graph clustering. *Proc. VLDB Endow.* **15**(11), 2908–2920 (2022). <https://doi.org/10.14778/3551793.3551840>
- Zhao, F., Agrawal, D., Abbadi, A.E., Metwally, A.: SpaceSaving± an optimal algorithm for frequency estimation and frequent items in the bounded deletion model. *Proc. VLDB Endow.* **15**(6), 1215–1227 (2022). <https://doi.org/10.14778/3514061.3514068>
- Zhao, F., Maiyya, S., Wiener, R., Agrawal, D., Abbadi, A.E.: KLL± Approximate quantile sketches over dynamic datasets. *Proc. VLDB Endow.* **14**(7), 1215–1227 (2021). <https://doi.org/10.14778/3450980.3450990>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.