MimoSketch: A Framework to Mine Item Frequency on Multiple Nodes with Sketches

Yuchen Xu
Peking University
xu.yuchen@pku.edu.cn

Wenfei Wu*
Peking University
wenfeiwu@pku.edu.cn

Bohan Zhao
Tsinghua University
zbh20@mails.tsinghua.edu.cn

Tong Yang
Peking University
yangtongemail@gmail.com

Yikai Zhao
Peking University
zyk@pku.edu.cn

ABSTRACT

We abstract a MIMO scenario in distributed data stream mining, where a stream of multiple items is mined by multiple nodes. We design a framework named MimoSketch for the MIMO-specific scenario, which improves the fundamental mining task of item frequency estimation. MimoSketch consists of an algorithm design and a policy to schedule items to nodes. MimoSketch's algorithm applies random counting to preserve a mathematically proven *unbiasedness* property, which makes it friendly to the aggregate query on multiple nodes; its memory layout is *dynamically* adaptive to the runtime item size distribution, which maximizes the estimation accuracy by storing more items. MimoSketch's scheduling policy balances items among nodes, avoiding nodes being overloaded or underloaded, which improves the overall mining accuracy. Our prototype and evaluation show that our algorithm can improve the item frequency estimation accuracy by an order of magnitude compared with the state-of-the-art solutions, and the scheduling policy further promotes the performance in MIMO scenarios.

CCS CONCEPTS

• **Information systems** → **Data stream mining**; *Data streams*; • **Theory of computation** → **Sketching and sampling**.

KEYWORDS

distributed data stream mining; unbiased sketch; scheduling policy

ACM Reference Format:
Yuchen Xu, Wenfei Wu, Bohan Zhao, Tong Yang, and Yikai Zhao. 2023. MimoSketch: A Framework to Mine Item Frequency on Multiple Nodes with Sketches. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599433

*Wenfei Wu is the corresponding author, and is with the School of Computer Science at Peking University.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599433

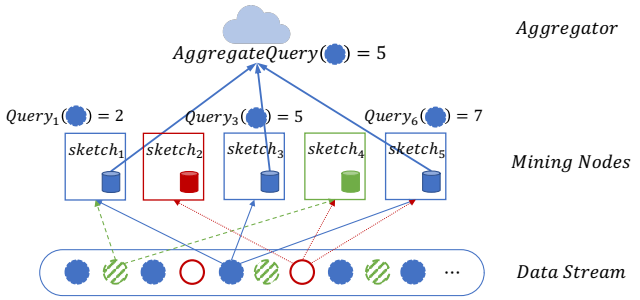


Figure 1: A schematic diagram of the MIMO scenario.

1 INTRODUCTION

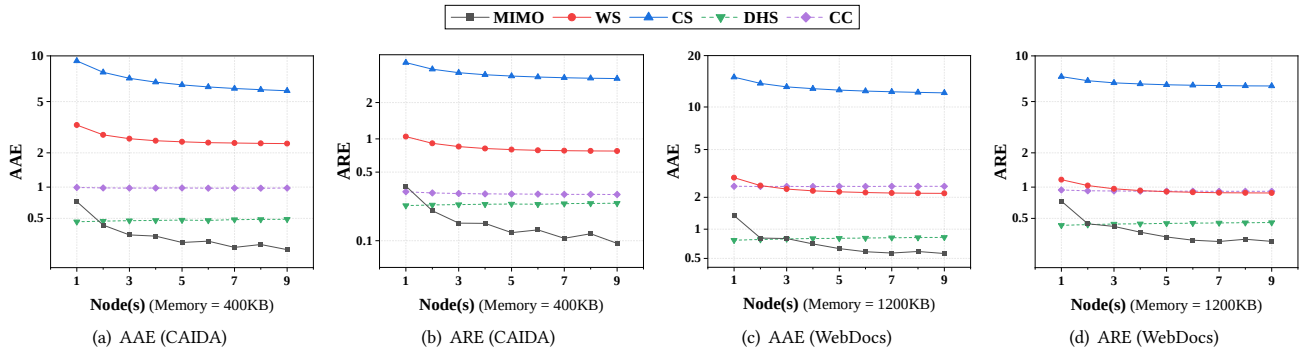
Sketches, are broadly investigated in the context of high-velocity data streams and limited memory resources [24, 25, 39, 40, 47, 49]. They are proven to provide accurate (approximate) item frequency estimation, a fundamental task in data stream mining; and demonstrate their effectiveness on other typical mining tasks: item size distribution estimation, heavy hitter detection, heavy change detection, and entropy estimation [4, 16, 18, 26, 43, 48].

For distributed data stream mining, we abstract a scenario where there are multiple items in a data stream and each item is mined consistently at multiple nodes, described in Figure 1. We name the setting of *Multiple Item mined on Multiple nOdes* as "MIMO"[1]. MIMO differs from traditional distributed scenarios where each partitioned data stream is mined by a single node, and then local results are aggregated (*e.g.*, by summation) for a global estimation. In the MIMO scenario, an aggregator aggregates data collected from all the candidate mining nodes (*e.g.*, by mean or median), which may gain an accuracy upgrade.

The MIMO abstraction occurs in numerous application scenarios, such as distributed data analytics [36], distributed machine learning [21], network telemetry [19], and IoT sensing [6], *etc.* We would further demonstrate MIMO's rationality and generality in various application use cases in Section 2.2. The MIMO pattern not only introduces redundancy for fault tolerance, but can also improve the mining accuracy to some extent (shown in Section 6.3.2).

Unfortunately, most existing sketch solutions do not show satisfactory performance in the MIMO scenario, even some applicable

[1]Coincidentally, such a scenario is also similar to the Multiple-Input-Multiple-Output (MIMO) antenna architecture in wireless networks, where each data stream is transmitted to multiple receivers simultaneously through multiple antennas.



(a) AAE (CAIDA) (b) ARE (CAIDA) (c) AAE (WebDocs) (d) ARE (WebDocs)

Figure 2: Unbiased MimoSketch outperforms baseline algorithms with more nodes.[2]

to traditional distributed scenarios [34, 41, 50]. There are two main reasons. First, most recent well-crafted sketch algorithms are *biased* [14, 32, 42, 46, 47], *i.e.*, lack of the mathematical guarantee of *unbiasedness*, which makes them unsuitable for distributed data stream mining. The unbiasedness means the expectation of each item's frequency estimation equal to its real occurrence. Without such theoretical property, the estimation error of biased sketches would be preserved or accumulated when multiple mining nodes' results are aggregated. There are few *unbiased* sketch algorithms proposed [7, 22, 24, 25]. However, they typically have larger theoretical variance and error bounds; thus, they show less accurate estimation than the biased ones empirically. Figure 2 shows an experimental example. As the number of mining nodes increases, the estimation error (AAE and ARE in the figure) of biased algorithms preserves (Cuckoo Counter [32]) or accumulates (DHS [47]); but that of unbiased ones (WavingSketch [25], Count Sketch [7], and MimoSketch in this paper) decreases in different convergence trends, taking advantage of the aggregate query on multiple nodes.

Second, when applying a sketch algorithm to multiple distributed nodes, little attention has been paid to the policy design yet. On the one hand, duplicating the stream to all nodes may cause most nodes overloaded, which could lead to non-optimal results [29]; on the other hand, making all streams recorded on a single node degrades to traditional single-node mining, which could not leverage the MIMO setting. The above two naïve methods are not acceptable, and a dedicated scheduling policy assigning items to nodes may further improve the performance in MIMO scenarios.

This paper presents a framework for item frequency estimation under the MIMO scenario, including the *algorithm* design of an unbiased sketch which can be deployed on each mining node and the *policy* design to schedule data streams to a set of nodes. The framework is named MimoSketch for its MIMO-friendly property. Both the algorithm and policy design aim to promote the accuracy of item frequency estimation.

MimoSketch organizes its memory space as an array of buckets and hashes each item to one for frequency counting. Inside each bucket, the memory layout is *dynamically* organized as several slots and a sharing counter in the runtime, with each slot adaptive to its items' frequency, *i.e.*, large space for hot (frequent) items and small space for cold (infrequent) ones. Thus, the limited memory can be maximally utilized to store more items, and the overall estimation accuracy is improved. On the basis of dynamic memory management, MimoSketch preserves the property of *unbiasedness* by randomly counting each item positively or negatively, *i.e.*, +1 or −1. Thus, a queried item's collided items are expected to be counted as 0, not affecting the queried one. Meanwhile, dynamic memory transformation is carefully designed to ensure each item in the same slot consistently moves (expands or expelled), avoiding being split into different slots and consequently underestimated.

MimoSketch formulates the scheduling policy as an optimization problem. It is a Mixed Integer Programming (MIP) problem, with the objective of minimizing the collision rate at the most overloaded node and the constraints of assigning each item to its candidate nodes. The MIP problem can be approximated by a Linear Programming (LP) problem with random rounding, which gives a near-optimal solution within a polynomial time. In this paper, we make the following contributions.

- We propose a framework MimoSketch to perform item frequency estimation in MIMO scenarios, including a sketch mechanism deployed on each node and a policy to schedule streams to distributed mining nodes.
- MimoSketch's algorithm offers unbiased and accurate estimation which suits for MIMO scenarios, and we also mathematically prove the unbiasedness of MimoSketch and provide its error bounds.
- MimoSketch's scheduling policy balances the mining workload of distributed nodes and improves overall accuracy.
- We conduct extensive experiments on both single-node and MIMO settings. Experimental results show that MimoSketch's algorithm outperforms unbiased sketches by 8.13 ∼ 38.27 times for a single node in terms of estimation accuracy, and MimoSketch's policy could further improve the performance by up to 169% than baselines for MIMO scenarios.

2 MOTIVATION AND BACKGROUND

In this section, we first elaborate on the problem formulation and application scenarios for MIMO. Then, we review prior work related to MimoSketch. Lastly, we highlight the goal, intuitions, and pitfalls of MimoSketch.

[2]We exploit all 1 ∼ 9 candidate nodes to mine the same data stream, respectively, and take the average as the aggregated result. Experimental workloads, metrics, and baselines are described in Section 6.1.

2.1 Problem Formulation

Let $\mathcal{S}$ be a data stream containing n items $e_1, \ldots, e_n$, adding up to N appearances. Each distinct item e_i can appear several times. Let f_i be e_i's frequency, then $f_i \geq 1$, and $\sum_{i=1}^{n} f_i = N$. Different items are identified by their unique IDs, and algorithms `Insert` and `Query` each e_i according to its ID. When the stream $\mathcal{S}$ ends, e_i's queried frequency $\hat{f}_i$ is an estimation of f_i.

In the MIMO setting, assume there are m nodes $n_1, \ldots, n_m$. Each item can be mined by a subset of distributed nodes, called its candidate nodes ($Cand_i$). When processing, multiple nodes `Insert` different items to its local mining digest independently. Querying the frequency estimation of an item is performed by `AggregateQuery`, an operation to `Query` the item's mining nodes ($Mine_i$) and to return the aggregated value of the results, *i.e.*, $\hat{f}_i = \underset{n_j \in Mine_i}{Aggr} \hat{f}_i^{(j)}$, where $\hat{f}_i^{(j)}$ represents the estimated occurrence of item e_i at node n_j and the aggregation method can be mean or median.

Besides, MIMO scenarios also need a *scheduling policy* to select each item's mining nodes from its candidate nodes ($Mine_i \subseteq Cand_i$). Carefully scheduling each item to its candidate nodes can improve the mining quality notably.

2.2 Application Scenarios

The MIMO scenario is an abstraction of many applications, *e.g.*, distributed data analytics, distributed machine learning, network telemetry, distributed IoT, *etc*, where our proposed solution MimoSketch can be applied to improve their performance.

Distributed data analytics. Distributed data analytic systems partition a dataset onto distributed servers (*e.g.*, HDFS), and there are several workers following a MapReduce scheme [10] to perform jobs. Sketches can be applied to data analytic jobs such as WordCount to get approximate but fast frequency estimation of words [36]. By allowing a partition mined by multiple workers with sketches and aggregating their query results, the overall query accuracy can be improved, which matches the MIMO abstraction.

Distributed machine learning. In data-parallel distributed model training, the dataset is partitioned to multiple workers. Iteratively, each worker trains the model locally to obtain a gradient, and all workers' gradients are aggregated (averaged) in the model update. In the Parameter Server (PS) system, one or multiple PSs perform the gradient aggregation. Researchers propose to apply sketches to compress the gradients [20, 21, 30], which reduces the system communication overhead but sacrifices precision. The MIMO abstraction can further improve the accuracy of these sketch-based gradient aggregation systems by assigning gradient partitions to PSs according to their processing capacity, and aggregating each partition's sketches from its assigned PS(s).

Network-wide telemetry. A network consists of multiple switches and hosts, with massive data streams traversing the switches between hosts. Network telemetry monitors multiple data streams on multiple devices, which assists the operator in network management. Sketches are demonstrated to be a powerful tool to measure network flows [19, 32]. By measuring a flow on one or several of its on-path switches, the flow estimation accuracy can be improved with the aggregation of sketches on the assigned switches, which matches the MIMO abstraction.

Multi-spot sensing in the Internet of Things (IoT). Nowadays, ubiquitous IoT devices are increasingly deployed. These devices individually sense the environment and collaboratively provide data stream statistics to abundant atop applications, such as anomaly detection [6]. One data stream (*e.g.*, temperature and humidity) can be measured by multiple devices in the same sensing range, and their results can be aggregated (averaged) to achieve an improved accuracy. This is a MIMO scenario where MimoSketch can be applied to promote measurement accuracy, eliminate noises/outliers, and consequently improve the application performance.

2.3 Related Work

Existing methods mainly design data structures to improve mining accuracy or efficiency, with less consideration for distributed scenarios, especially the MIMO scenario. Among the frequency estimation algorithms, the family of sketch-based algorithms are usually accurate and outperforms the counter-based ones [27, 28, 35] in terms of accuracy and memory efficiency. Thus, we mainly discuss sketch-based algorithms. In addition, the policy to schedule data streams among multiple mining nodes has not been widely discussed. Table 1 lists the feature comparisons between the representative algorithms and our solution MimoSketch.

2.3.1 Sketch-based Algorithms. As a probabilistic data structure, sketch has been extensively studied and applied in many data mining works [3, 7, 9, 11, 31, 37]. They can achieve $O(1)$ memory access by hash functions, *i.e.*, high throughput [41]; through complicated adaptive strategy, they can improve accuracy by orders of magnitude compared to counter-based ones, *i.e.*, high accuracy [32]. However, few sketch-based algorithms have elegant mathematical guarantees such as unbiasedness, which is a necessity for MIMO.

Unbiased sketches. As mentioned above, few sketches have the good theoretical property of unbiasedness. Count Sketch [7] is a basic unbiased sketch solution. It contains d arrays of buckets ($B_j[], j = 0, \ldots, d-1$). When an item e_i comes, it computes d pairwise-independent hash functions ($h_j(.)$) to index a bucket ($B_j[h_j(e_i)]$) and updates the corresponding counter by $s(e_i) \in \{1, -1\}$. The estimated frequency of e_i is the mean or median of $B_j[h_j(e_i)] \times s(e_i)$ ($j = 0, \ldots, d-1$). Its unbiasedness is intuitive, but accuracy is severely sacrificed due to a higher variance. WavingSketch [25], another sketch algorithm based on unbiased estimation, provides better accuracy than Count Sketch by filtering out hot and cold items within each bucket.

Biased sketches. There are also many sketch algorithms achieving superb accuracy but sacrificing unbiasedness [40–42, 46, 49]. To our best knowledge, two state-of-the-art sketch algorithms provide impressive accuracy, Cuckoo Counter [32] and DHS [47]. They both hash item IDs as shorter *fingerprints* to save storage and adapt data structures to item actual frequency for efficient space utilization. In addition, Cuckoo Counter uses partial-key cuckoo hashing [12] to partly address the hash collision problem; DHS dynamically adjusts the structure according to the actual item size distribution to store more items. However, due to hash collision or fingerprint collision, Cuckoo Counter always returns a value of overestimation; while DHS simply drops cold items when memory is tight, it does not have the property of either one-sided error or unbiasedness. In addition, these biased algorithms do not adapt to MIMO scenarios.

Table 1: Comparison of MimoSketch and the State-of-the-art Algorithms

Property \ Algorithm	MimoSketch	WavingSketch	Count Sketch	DHS	Cuckoo Counter
Unbiasedness	✓	✓[3]	✓	×	×
Hot/Cold Item Friendliness	Both	Hot	Hot	Both	Both
Frequency Estimation Accuracy	★★★★★	★★★	★★	★★★★★	★★★★
MIMO-scenario Elasticity	★★★★★	★★★★	★★★★★	★★	★★★

The rankings (the number of stars ★) on "Frequency Estimation Accuracy" and "MIMO-scenario Elasticity" are based on the experiments.

2.3.2 MIMO-scheduling Algorithms. A class of work discusses the scheduling policy for sketches in distributed data stream measurement in the network application [15, 38, 45], where each flow traverses multiple switches, in accordance with the MIMO scenario. For example, the scheme in [29] aims to reduce redundant statistics collected by sketches in the network: it selects the measurement node(s) in a topology by the graph coloring algorithm; CountMax [44] co-designs a sketch with a coordination algorithm, where it hashes an item to an ingress switch or egress switch. Different from these two works which try to avoid redundant counting, OmniMon [19] designs heterogeneous measurement methods on end hosts and switches, and deploys an instance on each node.

Differently, MimoSketch provides a new observation that (when the memory is not extremely tight) mining each item on multiple nodes with moderate redundancy could provide a better accuracy, and MimoSketch's policy in Section 5 tries to make a trade-off between per-node workload and the aggregate query's gain. In addition, MimoSketch specifically designs an unbiased sketch which is suitable for aggregate query in MIMO scenarios.

2.4 Goal, Intuitions, and Pitfalls

Goal and intuitions. Our goal is to design a mining framework providing accurate item frequency estimation to the best extent for the MIMO-specific scenario. The framework involves the design of a sketch algorithm on each node and a policy to schedule streams to nodes. Hopefully, the algorithm should be *unbiased* and *accurate* so that the frequency estimation of an item would be closer to its actual size in MIMO. The policy design should *balance* the use of multiple nodes, avoiding any node being overloaded or underloaded so as to maximize the overall accuracy.

The intuition for achieving unbiasedness is to apply the *random counting* method, *i.e.*, updating item frequency by +1 or −1 randomly. In sketch algorithms, a hash collision would cause two items to be counted in the same slot, and thus be overestimated. In MimoSketch, instead of simply counting an item occurrence as +1, one item's occurrence is counted as +1 or −1 (consistent among the item's all occurrences). MimoSketch uses a hash function to map each item's update to $+1/-1$. For collided items, the expectation of their total counts would be 0, meaning unbiased.

The intuition for improving the accuracy is to improve the *memory efficiency*: storing more items with limited memory. The random counting method increases the variance of the estimation, which is why traditional unbiased solutions have inferior performance (accuracy) than biased ones. MimoSketch adds two techniques to improve memory efficiency and complement the accuracy loss: (1) within each bucket, instead of using fixed-length slots (like existing unbiased algorithms [7, 25]) which wastes space for cold items, MimoSketch dynamically organizes the memory layout of the slots, adaptive to the item size characteristics (distribution). (2) Each slot is supposed to store an item ID to identify collisions and record item frequency. MimoSketch compresses the item ID as a shorter fingerprint, also saving storage space. As the stream proceeds, hot items would eventually evict cold items into the sharing counter without fingerprints; eventually, a cold item would seldom be falsely recognized as hot (except only when their fingerprints collide).

The intuition for balancing the mining workload is to *schedule streams to multiple nodes*. MimoSketch formulates a mixed integer programming (MIP) model. The constraints are that (1) each stream can be mined by its candidate nodes and (2) each stream is mined at least a few times (avoiding underload). The objective is to minimize the workload of the most loaded node (avoiding overload).

Pitfalls and approaches. The synthetic framework of the methods above should be carefully devised; otherwise, the dynamic memory organization could violate the unbiasedness requirements. There are two subtle pitfalls.

First, dynamic memory organization involves expelling cold items when adaptively allocating more space to hot ones. Dropping cold items would incur bias. Thus, an extra counter is needed for these expelled items, and it should also be unbiased.

Second, dynamic memory organization involves expanding the space of slots with hot items. But a slot could contain several fingerprint-collided items, and the expansion should never split an item into different locations, which would cause the eventual counting to be partial. Thus, items belonging to a slot with a fixed-length fingerprint would be consistently stored and counted at the same place in all cases of memory re-organization, whether the slot expands, stays unchanged, or is expelled to the sharing counter.

3 MIMOSKETCH'S ALGORITHM

Section 3.1 presents MimoSketch's data structure. Section 3.2 and Section 3.3 describe MimoSketch's workflow on a single node and distributed nodes, respectively. Table 2 lists all the symbols and their meanings in the algorithm description.

3.1 Data Structure

MimoSketch organizes the memory as an array of l buckets. Each bucket contains two parts, a sharing counter and d slots, attached with a metadata recording the layout inside the bucket, such as

[3]WavingSketch is partly unbiased because its top-k query cannot prove the unbiasedness [25].

Table 2: List of Symbols to Use and their Meanings

Notation	Meaning
$\mathcal{S}$	a data stream
e_i	i_{th} distinct item in $\mathcal{S}$
l	number of buckets in MimoSketch
d	number of slots in a bucket
k	number of levels within slots
$B[i]$	i_{th} bucket of MimoSketch
$B[i].count$	sharing counter of $B[i]$
$B[i].f[e_i]$	slot where e_i is stored
$B[i].f[e_i].fp$	*fingerprint* stored in $B[i].f[e_i]$
$B[i].f[e_i].fq$	*frequency* stored in $B[i].f[e_i]$
$h(.)$	hash function mapping items to $\{0, \cdots, l-1\}$
$s(.)$	hash function mapping items to $\{1, -1\}$
$fp(.)$	hash function computing *fingerprint* of an item

levels and boundaries of slots. Each slot is a key-value pair of <*fingerprint*, *frequency*>, where the *fingerprint* records an identification based on item ID, and the *frequency* records items' estimated size.

Slots in a bucket can be hierarchically adjusted into k different levels, and the *fingerprints* in each level have the *same* V bits. The lengths of the *frequency* field are different, ranging from W_1 to W_k bits. With a U-bit sharing counter, a T-bit metadata, and N_i slots for each level-i, the total space for a bucket is $\sum_{i=1}^{k} N_i \cdot (W_i + V) + U + T$ bits. In each bucket, slots can dynamically adapt to the item characteristics; that is, a bucket starts with many level-1 slots, each with the minimum space, and gradually increases to fewer higher-level slots, each with more space to accommodate hot items.

3.2 MimoSketch on a Single Node

3.2.1 Initialization. Figure 3 shows the data structure of MimoSketch. At the beginning of stream processing, all fields in MimoSketch are initialized to zero, and each slot is at the lowest level-1.

3.2.2 Insertion. Algorithm 1 in Appendix A shows the pseudocode of the insertion workflow of MimoSketch. When a data stream comes, each item e_i is mapped to the bucket $B[h(e_i)]$ through one hash function $h(.)$. All slots within the bucket are iterated to find one with a matched fingerprint. MimoSketch computes another hash function $s(.)$ to decide whether to count the item's occurrence as +1 or −1, *i.e.*, increasing or decreasing one to the item's frequency field. There are the following three cases:

Case 1: If e_i's fingerprint is already recorded in a certain slot, MimoSketch updates the slot frequency by $s(e_i)$ (*i.e.*, +1 or −1). However, since slot sizes are limited, the slot may overflow. In such a circumstance, MimoSketch makes three attempts.

Case 1.1: MimoSketch attempts to find whether any higher-level slot's frequency is less than the capacity of e_i's slot in the same bucket. If there is one, MimoSketch swaps the contents of the two slots. The overflow problem is addressed without cost.

Case 1.2: If the first attempt fails, MimoSketch will conduct a *LevelUp* operation. It first tries to put e_i into an empty higher-level slot if exists. If not, MimoSketch will adjust the $B[h(e_i)]$'s structure, making room for level-$(t+1)$ slots (assuming $B[h(e_i)].f[e_i]$'s level is t now). MimoSketch iterates on levels from level-1 to level-t to find one level with enough slots to deallocate for new slots.

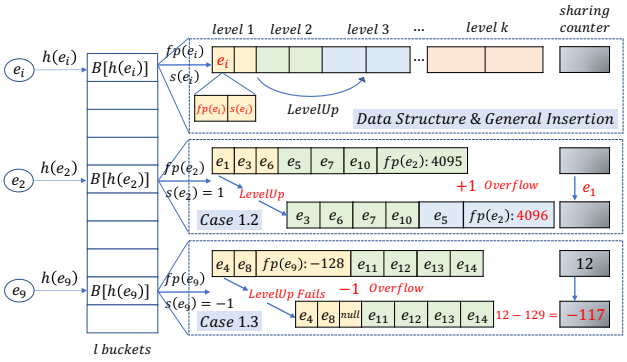


Figure 3: Data structure and workflow of MimoSketch.

For example, if *fingerprint* occupies 8 bits, $d = 8$, $k = 3$, *frequency* field for each level is 8, 12, and 16 bits, respectively, the *LevelUp* operation could reallocate five level-1 slots for four level-2 slots for $5 \times (8+8) = 4 \times (8+12)$, or three level-1 ones for two level-3 ones for $3 \times (8+8) = 2 \times (8+16)$.

If the attempt succeeds, the *LevelUp* will create n_2 level-$(t+1)$ slots from n_1 lower-level slots. Then $(n_1 - n_2)$ of the lower-level slots with minimum frequency are expelled to the sharing counter, and *all* the other old slots are moved towards the same or higher-level slots. The expelled items cannot be reallocated with new empty slots in later processing for unbiasedness (further explained in Case 2). The slot boundaries are adjusted to keep the memory compact, which is recorded in the metadata.

Case 1.3: If the aforementioned two attempts fail, indicating no room to address the overflow, MimoSketch straightforwardly clears e_i's slot and evicts it to the sharing counter. Moreover, the status is set $false$, meaning the slot is "deprecated". Note that a deprecated slot is not cleared, and its fingerprint is kept: if the deprecated slot's item appears in the future, MimoSketch could tell that the item is stored in the sharing counter and would not allocate a new slot for the item (*i.e.*, Case 2), not incurring bias.

Case 2: If e_i's fingerprint is not found but there is an empty usable slot in $B[h(e_i)]$, MimoSketch fills in the slot with $<fp(e_i), s(e_i)>$.

It is worth noting that once an item is expelled from a slot to the sharing counter, it should never be allocated to a new slot; otherwise, the item is counted in two places, hurting unbiasedness. Case 1.3 circumvents this issue by deprecating the slot and keeping the fingerprint. In Case 1.2, MimoSketch adds a flag indicating no reallocation in the bucket metadata: once the eviction in Case 1.2 happens, MimoSketch sets the flag and disables Case 2 (in this case, the algorithm proceeds to Case 3).

Case 3: If the former two cases can not complete the insertion, the sharing counter will handle the current item. MimoSketch updates the sharing counter by $s(e_i)$, which also complies with the unbiasedness guarantee.

3.2.3 Query. When querying an item e_i's estimated frequency, MimoSketch first computes $h(e_i)$ and $fp(e_i)$, and then locates a bucket $B[h(e_i)]$ and iterates slots in the bucket to match the fingerprint.

If $fp(e_i)$ matches a slot's fingerprint $B[h(e_i)].f[e_i].fp$, `Query` will return the estimated frequency $(B[h(e_i)].f[e_i].fq) \times s(e_i)$. Otherwise, MimoSketch reads the sharing counter and returns its value $B[h(e_i)].count \times s(e_i)$ as the answer to the query.

Note that if the frequency of an item e_i's slot or sharing counter has a contrary sign to $s(e_i)$, we can assert that fingerprint collision must have happened, and multiple items are updated in the same frequency field or counter. Because negative estimated frequencies contradict the fact of data stream processing, we can thus return those estimated frequencies as zero in practice.

3.2.4 Examples. Figure 3 shows three examples of insertion. The general e_i's insertion has two steps. MimoSketch first computes $h(e_i)$ to locate the bucket and then computes $fp(e_i)$ and $s(e_i)$ to locate the slot and decide update direction (+1 or −1).

The insertion of e_2 is an example of Case 1.2. After the update, the level-2 slot for e_2 is to overflow, and there are no higher-level slots for swapping, then MimoSketch successfully operates *LevelUp* by deallocating three level-1 slots to create two level-3 ones. e_2 can fit in a level-3 slot, other items are also shifted, and the victim e_1 is expelled to the sharing counter. Then the flag in metadata is set, disabling Case 2.

The insertion of e_9 is an example of Case 1.3. Because all the slots in the bucket are already occupied, and there are not enough level-1 slots to deallocate (at least 5 level-1 ones for 4 level-2), *Exchange* and *LevelUp* attempts fail. Owing to e_9's overflow, it is expelled to the sharing counter, with the slot frequency accumulated to the sharing counter. The original slot's location is marked as *false* to deprecate the slot, and the fingerprint remains there.

For queries, the result of $\texttt{Query}(e_2)$ is $4096 \times (+1) = 4096$, retrieved by iterating the slots to find a matched fingerprint. That of e_9 is $(-117) \times (-1) = 117$, because there is no matched fingerprint and thus returns the frequency in the sharing counter.

3.3 MimoSketch on Distributed Nodes

In distributed data stream mining, each node installs an instance of MimoSketch. The hash functions $h(.)$, $s(.)$, and $fp(.)$ differ among nodes, which could introduce randomness and reduce the estimation errors when aggregating multi-node queries. In the MIMO scenario, MimoSketch applies a scheduling policy to assign which nodes to monitor each item, which is discussed in Section 5.

During processing, a data stream can be mined by a subset of nodes, and the items are recorded by the assigned nodes. The insertion operation on each mining node is the same as that in the single-node MimoSketch's algorithm.

As for item query, MimoSketch queries all mining nodes of the item and aggregates the queried results. The final return is the aggregate query result, which can be the mean or median value of the results from all mining node(s). For example, if aggregating by average, *i.e.*, $\frac{1}{m}\sum_{j=1}^{m}\texttt{Query}_j(e_i)$, when there are three nodes with $B_1[h_1(e_i)].f[e_i].fq = 5$, $s_1(e_i) = 1$, $B_2[h_2(e_i)].count = -3$, $s_2(e_i) = -1$, and $B_3[h_3(e_i)].f[e_i].fq = 7$, $s_3(e_i) = 1$, the final result of $\texttt{AggregateQuery}(e_i)$ is $(5 \times 1 + (-3) \times (-1) + 7 \times 1)/3 = 5$.

The mean taken in the aggregate query is a simple but reasonable estimator, as the expectation is unbiased. As the scale of distributed nodes increases, the aggregate query is likely to converge to the unbiased expectation with lower variance, and thus achieves superb accuracy. And the median aggregation behaves better in experiments (the same as [7]), for the median is more robust to outliers.

4 THEORETICAL PROPERTIES

In this section, we show the theoretical properties of MimoSketch, including unbiasedness, variance, and error bounds.

4.1 Unbiasedness

MimoSketch's unbiasedness property is defined as that for each item e_i inserted, the frequency estimation $\hat{f}_i$ is unbiased, equal to its real frequency f_i. The intuition comes from Count Sketch [7]. For an item e_i and its collided item $e_j (i < j)$, if their counting functions $s(e_i)$ and $s(e_j)$ are independent, we have $E(s(e_i) \cdot s(e_j)) = 0$. And thus, as the item e_i is counted in its slot/counter, its collided items would accumulate the frequency with an expectation of 0, *i.e.*, $E(\sum_{e_j<e_i} f_j \cdot s(e_j) \cdot s(e_i)) = 0$. We give the formal proof in Appendix B.1.

4.2 Variance and Error Bounds

We show MimoSketch's variance and error bounds of `Query` and analysis of `AggregateQuery`. Detailed proofs are given in Appendix B.2.

Variance. *Let $e_1, e_2, \ldots, e_n$ be the items hashed into $B[h(e_i)]$, the bound of the variance of MimoSketch's estimation is that*

$$Var\left[\hat{f}_i\right] \le \sum_{e_j<e_i} (f_j)^2.$$

Error bound based on $\|f\|_2$. *Let $l = \frac{e}{\epsilon^2}$, where l is the total number of buckets in MimoSketch, and e is the total number of distinct items. Based on 2-norm $\|f\|_2$ (i.e., $\sqrt{\sum_{e_i \in \mathcal{S}} (f_i)^2}$),*

$$P\left[\left|\hat{f}_i - f_i\right| \ge \epsilon \|f\|_2\right] \le \frac{1}{e}.$$

Error bound based on $\|f\|_1$. *Let $l = \frac{e}{\epsilon}$, $\|f\|_1 = \sum_{e_i \in \mathcal{S}} |f_i|$,*

$$P\left[\left|\hat{f}_i - f_i\right| \ge \epsilon \|f\|_1\right] \le \frac{1}{e}.$$

AggregateQuery's accuracy. Considering the mean aggregation, by the law of large numbers [17], an item's `AggregateQuery` would return the item frequency estimation with an unbiased expectation and a lower variance.

Assume there are m distributed mining nodes, and the j_{th} node returns an estimation $\hat{f}_i^{(j)}$. Then

$$E\left[\frac{1}{m}\sum_{j=1}^{m}\hat{f}_i^{(j)}\right] = \frac{1}{m}\sum_{j=1}^{m} E\left[\hat{f}_i^{(j)}\right] = 0,$$

and

$$Var\left[\frac{1}{m}\sum_{j=1}^{m}\hat{f}_i^{(j)}\right] = \frac{1}{m^2} Var\left[\sum_{j=1}^{m}\hat{f}_i^{(j)}\right] \le \frac{1}{m}\sum_{e_j<e_i} (f_j)^2. \quad (1)$$

5 MIMOSKETCH'S SCHEDULING POLICY

This section builds the model of *scheduling policy* to optimize the utilization of multiple mining nodes in a MIMO-modeled system.

Problem statement. In the MIMO scenario, an item may be in the measurement range of a subset of nodes. Each subset is the *candidate*

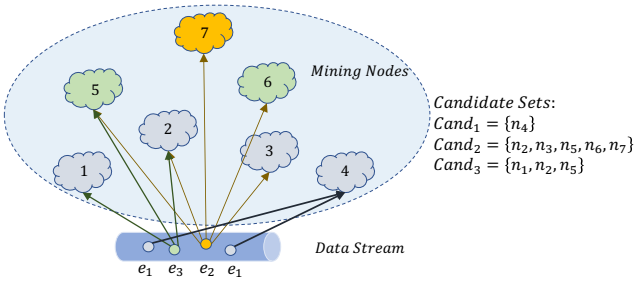


Figure 4: An example MIMO scenario.

set of each item inside a data stream. An item can be recorded by one or several nodes in its candidate set. Deciding the mining nodes of each item is critical to the overall accuracy: if an item is measured by too few nodes, its frequency estimation accuracy cannot gain the benefit of the aggregate query (see Equation (1)); if too many items are monitored by too many nodes, nodes would be overloaded and items collide on nodes, decreasing per-node accuracy. MimoSketch designs a policy to schedule items to its mining nodes, which improves the overall accuracy.

Figure 4 shows an example of a MIMO scenario, which is common in various applications, *e.g.*, big data, IoT and networks. There are seven mining nodes in all. Each item can use 1, 3, or 5 nodes as candidate nodes. For example, node n_4 is responsible for item e_1, and item e_2 can be mined by five nodes n_2, n_3, n_5, n_6, and n_7.

Modeling. We use notations in Table 3 in the optimization problem modeling. Although there are several accuracy metrics for sketches, its inaccuracy comes from items' hash collisions in essence, and the probability of hash collision on a node is positively correlated to its mining load ($\sum_i x_{ij}$). Thus, an unbalanced load among nodes would cause the overloaded nodes to have more collisions. Therefore, the objective function of MimoSketch's policy prefers to reduce the load of the most loaded node to achieve load balancing, *i.e.*, Equation (2).

As for the constraints, first, each item can only be mined at its candidate set (Equation (3)); second, each item should be monitored by a moderate amount of nodes (Equation (4)); and finally, whether a node monitors an item is binary (Equation (5)).

$$\text{minimize} \quad \max_j \sum_{i=1}^{\tilde{m}} x_{ij}. \tag{2}$$

$$\text{s.t.} \quad x_{ij} = 0, \forall j \notin Cand_i, \forall i, \tag{3}$$

$$\sum_{j=1}^{\tilde{n}} x_{ij} = C_i, \forall i, \tag{4}$$

$$x_{ij} \in \{0, 1\}, \forall i, j. \tag{5}$$

Discussions. *Approximation.* The max-min problem above (Equation (2) ~ (5)) is a Mixed Integer Programming (MIP) problem, which can be solved directly by solvers. When the problem scale grows drastically large, Equation (5) can be relaxed to a linear constraint,

$$0 \le x_{ij} \le 1, \forall i, j. \tag{6}$$

The problem then becomes a Linear Programming (LP) problem (Equation (2)~(4), and (6)). The LP solution gives x_{ij} in the real number domain, and we further round it randomly: rounding to 1 with the probability of x_{ij} and 0 otherwise. Theoretically, the LP approximation gives a result whose expectation equals the optimal MIP result [33], *i.e.*, E(Random-Rounded LP's Result) = MIP's Result.

Online algorithm. The policy algorithm is offline with the prior knowledge of data streams' candidate sets. This is a reasonable assumption in many scenarios, *e.g.*, multiple workers or mappers in big data, switches on the flow paths in a network, and IoT devices inside a sensing range. If there is no prior knowledge of data stream items, the algorithm can be periodically executed, each period based on the stream statistics from the previous period. Designing an online algorithm is our future work.

Table 3: List of Symbols for Modeling and their Meanings

Notation	Type	Meaning
m	const	number of nodes in the system
n	const	number of distinct items
x_{ij}	variable	whether e_i is monitored on node n_j
$Cand_i$	const	item e_i's candidate set
C_i	const	number of item e_i's mining nodes

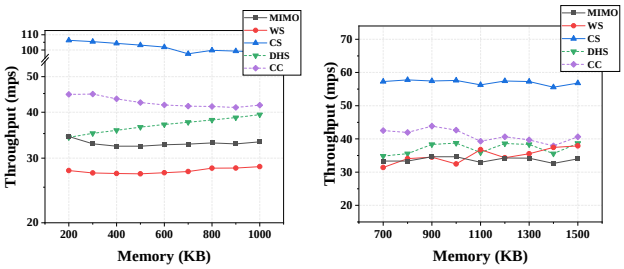
6 EVALUATION

This section shows extensive experimental results and demonstrates MimoSketch's excellent performance compared with baselines.

- MimoSketch's efficient memory utilization improves the mining accuracy by at least one order of magnitude compared to other unbiased algorithms, catching up or surpassing that of biased ones (Section 6.2).
- MimoSketch's counting method achieves unbiasedness, and in the MIMO scenario, the `AggreateQuery` with multiple MimoSketch instances benefits from the unbiasedness, outperforming all the baselines (Section 6.3.1).
- MimoSketch's scheduling policy makes better use of the global resources, which further improves overall frequency estimation accuracy with little solving cost (Section 6.3.2).

6.1 Experiment Setup

Details of the experiment settings are in Appendix C and our codes are open-sourced [2]. We perform experiments on 3.6GHz CPU with 32GB memory. We use CAIDA [5] and WebDocs [13] as datasets. We compare MimoSketch (MIMO) with four baselines: Count Sketch (CS) [7] and WavingSketch (WS) [25] to represent unbiased sketches, Dynamic Hierarchical Sketch (DHS) [47] and Cuckoo Counter (CC) [32] to represent biased ones. These algorithms may be mentioned in abbreviations for convenience. All algorithms are implemented in C++ and open-sourced anonymously at GitHub [2], and their parameters are tuned for the best performance. We evaluate each algorithm's performance on the item frequency estimation task, with metrics of Average Absolute Error (AAE) and Average Relative Error (ARE) indicating estimation error, and throughput indicating efficiency. The formal definitions of the task, metrics, and parameter settings are also in Appendix C.



(a) CAIDA (b) WebDocs

Figure 5: Throughput of insertion on CAIDA and WebDocs, varying memory.

6.2 Performance on a Single Node

First, Figure 5 shows the throughput of five sketches. Except that Count Sketch has much faster speed (at the expense of poor accuracy), MimoSketch and the other three algorithms' throughput are all satisfactory, around 25 ∼ 45 Mps.

As shown in Figure 6, the AAE and ARE of MimoSketch could not only outperform unbiased algorithms, but catch up or exceed biased but accurate adaptive methods under some circumstances. For the CAIDA dataset, when memory varies from 200KB to 1000KB, the AAE of MimoSketch is at most 8.13, 28.98, 1.03, 1.70 times lower than that of WS, CS, DHS, and CC, respectively; the ARE of MimoSketch is at most 10.78, 38.27, 1.39, 1.98 times lower, respectively. We can observe that MimoSketch has a faster convergence speed and finally outperforms most algorithms, which is quite impressive because previous unbiased algorithms almost lag behind well-designed biased ones by at least one order of magnitude. MimoSketch achieves such accuracy while keeping the unbiased property, benefiting from its dynamic adaptive memory organization.

6.3 Performance on Distributed Nodes

MimoSketch's unbiasedness makes it more applicable in distributed data stream mining, the MIMO scenario particularly. To test the improvement of the MimoSketch's algorithm and policy in the MIMO scenario, we conduct extensive experiments on a prototype whose detailed settings are shown in Appendix C.

6.3.1 Comparison of Sketches in the MIMO Scenario. The experiment in Figure 2 already confirms that MimoSketch on multiple devices could leverage the law of large numbers [17], and therefore reduce variance and gain better accuracy. Note that in distributed settings, "memory" refers to space usage of each node.

In Figure 7, when the candidate set of an item e_i contains 3 or 5 nodes, we set C_i to 2. We apply the same MimoSketch's scheduling policy but install different sketch algorithms on nodes. MimoSketch could perform better than all other algorithms when integrated with the policy, at most outperforming WS, CS, DHS, and CC by 13.28, 24.95, 1.31, 1.55 times in CAIDA, and 12.33, 17.41, 1.45, 1.71 times in WebDocs.

6.3.2 Impact of MimoSketch's Policy. We then testify to the effectiveness of the MimoSketch's scheduling policy.

Table 4: Load-Balancing Performance of MimoSketch's Policy

Variation	random	MimoSketch
CAIDA	2.11e+10	3.97e+9
WebDocs	1.67e+10	3.08e+9

First of all, considering solving efficiency, our prototype spends 6.1 ∼ 9.9 seconds in each MimoSketch's policy experiment, where there are seven mining nodes and up to 1.8M item occurrences, indicating the fast solving velocity and low delay cost.

Then we take the variation of workloads on different nodes ($Var_j(\sum_i x_{ij})$) as a metric to evaluate the load-balancing performance of MimoSketch's policy compared to the random policy. Both policies set C_i to 3, and the random policy randomly chooses the same amount of nodes for each item. Table 4 shows a 5.32 5.42 times lower variation of MimoSketch's policy than that of random policy, proving its more balanced performance. Hence the policy achieves the target of balancing workloads.

Figure 8 compares MimoSketch's policy and naïve policies, where "1 node" means randomly choosing one node enabled for an item, and "5 nodes" means choosing all the candidate nodes for an item. We observe that MimoSketch's policy compared to the one-node policy has up to 100% accuracy improvement, and 169% compared to the five-node policy. Note that the one-node policy could outperform MimoSketch when the memory is extremely tight, because multi-node policies increase per-node hash collisions in this case, which is the dominant factor, but all three solutions show impractical large errors. When memory is not so tight, multi-node policies show more advantages than one-node policy, and MimoSketch further outperforms the five-node policy due to its scheduling design.

Figure 9 shows the accuracy of the framework with MimoSketch's policy and random policy. Both policies have the same parameters, *i.e.*, C_i is 2 or 3. MimoSketch's policy outperforms the random policy by up to 74%.

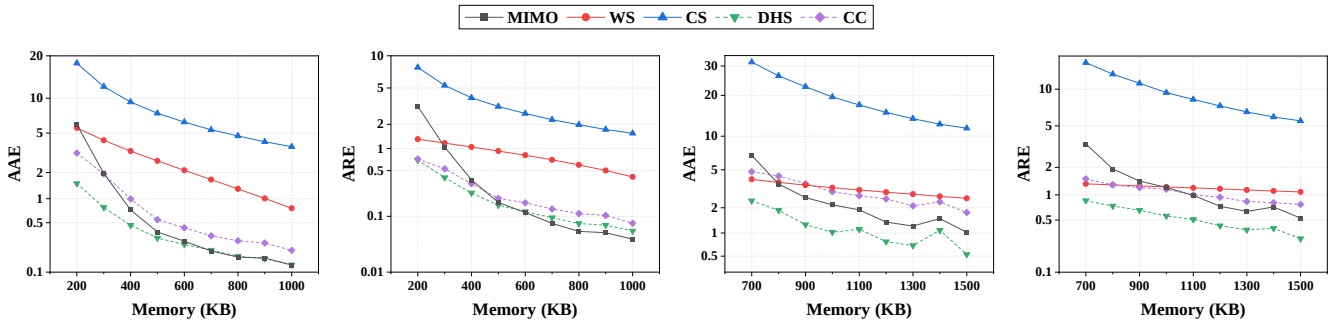
Experimental results manifest that, three mining nodes with the median query achieve the most accurate result in our prototype. We have proved the effectiveness of MimoSketch's scheduling policy.

7 CONCLUSION

MimoSketch is a framework for item frequency estimation in distributed data stream mining, especially the MIMO scenario. MimoSketch consists of a sketch and a policy. MimoSketch's algorithm applies the random counting method to preserve the unbiasedness and item size self-adaptive dynamic memory organization to promote accuracy. MimoSketch's policy balances multi-item mining workloads to multiple nodes, which improves overall accuracy. We evaluate MimoSketch on the item frequency mining task with CAIDA and WebDocs datasets. Experiments show that MimoSketch can achieve an orders-of-magnitude lower error rate of item frequency estimation, compared to Count Sketch, WavingSketch, DHS, and Cuckoo Counter.

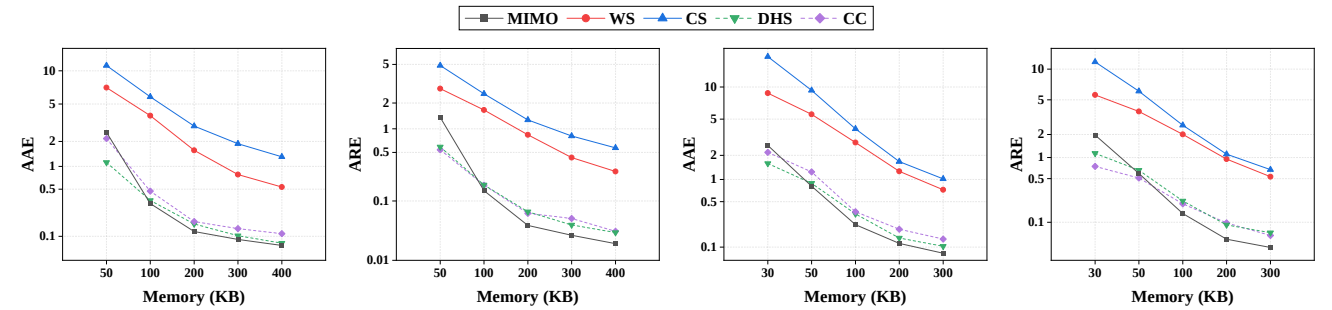
ACKNOWLEDGMENTS

Wenfei Wu is funded by the Faculty Startup Funding from the School of Computer Science at Peking University.



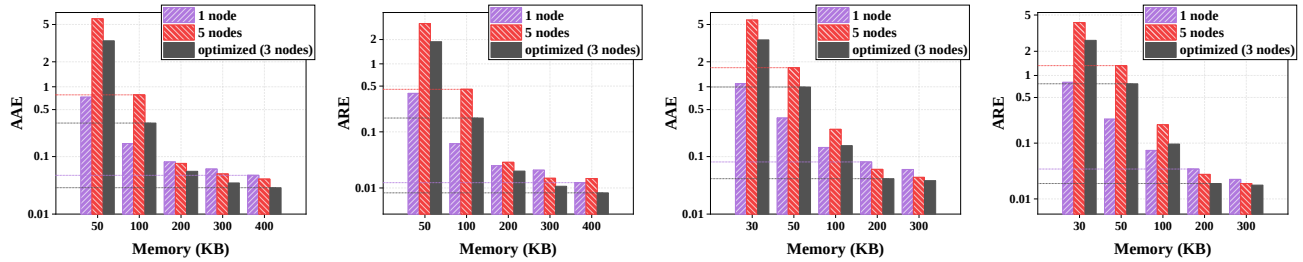
(a) AAE (CAIDA) (b) ARE (CAIDA) (c) AAE (WebDocs) (d) ARE (WebDocs)

Figure 6: Accuracy of item frequency estimation on CAIDA and WebDocs, varying memory.



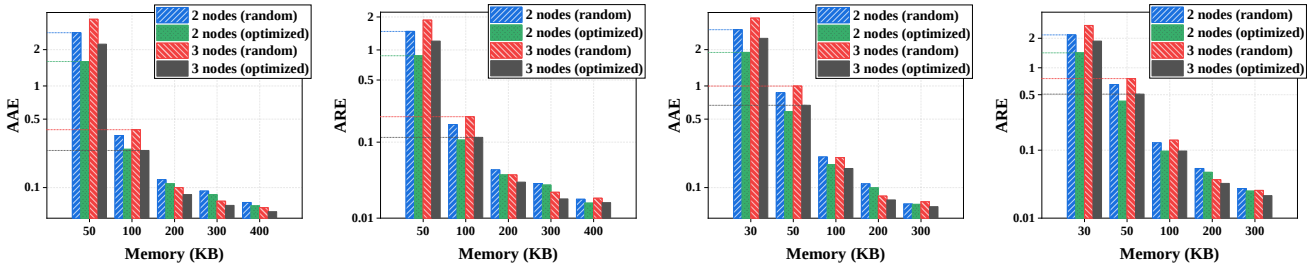
(a) AAE (CAIDA) (b) ARE (CAIDA) (c) AAE (WebDocs) (d) ARE (WebDocs)

Figure 7: Accuracy of distributed sketches with MimoSketch's policy on CAIDA and WebDocs, varying memory.



(a) AAE (CAIDA) (b) ARE (CAIDA) (c) AAE (WebDocs) (d) ARE (WebDocs)

Figure 8: Accuracy of the framework with MimoSketch's policy and naïve policies on CAIDA and WebDocs, varying memory.



(a) AAE (CAIDA) (b) ARE (CAIDA) (c) AAE (WebDocs) (d) ARE (WebDocs)

Figure 9: Accuracy of the framework with MimoSketch's policy and random policy on CAIDA and WebDocs, varying memory.

REFERENCES

[1] aappleby. 2015. MurmurHash. https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp

[2] Anonymous Author(s). 2022. Source codes of MimoSketch and other baselines. https://github.com/MimoSketch/MimoSketch

[3] MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni. 2018. Optimal distributed submodular optimization via sketching. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1138–1147.

[4] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. 2001. Towards Sensor Database Systems. In *Proceedings of the Second International Conference on Mobile Data Management*. 3–14.

[5] CAIDA. 2016. Anonymized Internet Traces. http://www.caida.org/data/overview

[6] Francesco Cauteruccio, Luca Cinelli, Enrico Corradini, Giorgio Terracina, Domenico Ursino, Luca Virgili, Claudio Savaglio, Antonio Liotta, and Giancarlo Fortino. 2021. A framework for anomaly detection and classification in Multiple IoT scenarios. *Future Generation Computer Systems* 114 (2021), 322–335.

[7] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding Frequent Items in Data Streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*. 693–703.

[8] Lucchese Claudio, Orlando Salvatore, Perego Raffaele, and Silvestri Fabrizio. 2003. WebDocs: a real-life huge transactional dataset. http://fimi.uantwerpen.be/data/webdocs.pdf

[9] Graham Cormode and S Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[10] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[11] Cristian Estan and George Varghese. 2003. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)* 21, 3 (2003), 270–313.

[12] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 75–88.

[13] FIMI. 2003. Frequent Itemset Mining Dataset Repository. http://fimi.uantwerpen.be/data/

[14] Xiangyang Gou, Long He, Yinda Zhang, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. 2020. Sliding sketches: A framework using time zones for data stream processing in sliding windows. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1015–1025.

[15] Jiqing Gu, Chao Song, Haipeng Dai, Lei Shi, Jinqiu Wu, and Li Lu. 2022. ACM: Accuracy-Aware Collaborative Monitoring for Software-Defined Network-Wide Measurement. *Sensors* 22, 20 (2022), 7932.

[16] Şule Gündüz and M Tamer Özsu. 2003. A web page prediction model based on click-stream tree representation of user behavior. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 535–540.

[17] Pao-Lu Hsu and Herbert Robbins. 1947. Complete convergence and the law of large numbers. *Proceedings of the national academy of sciences* 33, 2 (1947), 25–31.

[18] Qun Huang and Patrick PC Lee. 2014. Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 1420–1428.

[19] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 404–421.

[20] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. 2019. Communication-efficient distributed SGD with sketching. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 13142–13152.

[21] Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. 2018. Sketchml: Accelerating distributed machine learning with data sketches. In *Proceedings of the 2018 International Conference on Management of Data*. 1269–1284.

[22] Lu Jie, Chen Hongchang, Sun Penghao, Hu Tao, and Zhang Zhen. 2021. OrderSketch:: An Unbiased and Fast Sketch for Frequency Estimation of Data Streams. *Computer Networks* 201 (2021), 108563.

[23] Brian W Kernighan and Dennis M Ritchie. 1978. The C Programming Language, Prentice Hall. *Englewood Cliffs, New Jersey* (1978).

[24] Haoyu Li, Qizhi Chen, Yixin Zhang, Tong Yang, and Bin Cui. 2022. Stingy sketch: a sketch framework for accurate and fast frequency estimation. *Proceedings of the VLDB Endowment* 15, 7 (2022), 1426–1438.

[25] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. 2020. Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1574–1584.

[26] Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Iannaccone, and Anukool Lakhina. 2006. Detection and identification of network anomalies using sketch subspaces. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 147–152.

[27] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*. 346–357.

[28] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th international conference on Database Theory*. 398–412.

[29] Cheng-Chieh Peng, Kuo-Shiang Hsu, and Pi-Chung Wang. 2021. Collaborative Traffic Measurement Using Sketches for Software Defined Networks. In *2021 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*. IEEE, 81–87.

[30] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. 2020. FetchSGD: communication-efficient federated learning with sketching. In *Proceedings of the 37th International Conference on Machine Learning*. 8253–8265.

[31] Pratanu Roy, Arijit Khan, and Gustavo Alonso. 2016. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*. 1449–1463.

[32] Qilong Shi, Yuchen Xu, Jiuhua Qi, Wenjun Li, Tong Yang, Yang Xu, and Yi Wang. 2023. Cuckoo Counter: Adaptive Structure of Counters for Accurate Frequency and Top-k Estimation. *IEEE/ACM Transactions on Networking* (2023).

[33] Aravind Srinivasan. 1999. Approximation algorithms via randomized rounding: a survey. *Series in Advanced Topics in Mathematics, Polish Scientific Publishers PWN* (1999), 9–71.

[34] Lu Tang, Qun Huang, and Patrick PC Lee. 2019. MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2026–2034.

[35] Daniel Ting. 2018. Data sketches for disaggregated subset sum and frequent item estimation. In *Proceedings of the 2018 International Conference on Management of Data*. 1129–1140.

[36] Daniel Ting, Jonathan Malkin, and Lee Rhodes. 2020. Data sketching for real time analytics: Theory and practice. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3567–3568.

[37] Pinghui Wang, Yiyan Qi, Yuanming Zhang, Qiaozhu Zhai, Chenxu Wang, John CS Lui, and Xiaohong Guan. 2019. A memory-efficient sketch method for estimating high similarities in streaming sets. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 25–33.

[38] Hongli Xu, Shigang Chen, Qianpiao Ma, and Liusheng Huang. 2019. Lightweight Flow Distribution for Collaborative Traffic Measurement in Software Defined Networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1108–1116.

[39] Tong Yang, Siang Gao, Zhouyi Sun, Yufei Wang, Yulong Shen, and Xiaoming Li. 2019. Diamond sketch: Accurate per-flow measurement for big streaming data. *IEEE Transactions on Parallel and Distributed Systems* 30, 12 (2019), 2650–2662.

[40] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. 2018. HeavyGuardian: Separate and guard hot items in data streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2584–2593.

[41] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 561–575.

[42] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. 2017. Pyramid sketch: A sketch framework for frequency estimation of data streams. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1442–1453.

[43] Shanshan Ying, Flip Korn, Barna Saha, and Divesh Srivastava. 2015. Treescope: finding structural anomalies in semi-structured data. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1904–1907.

[44] Xiwen Yu, Hongli Xu, Da Yao, Haibo Wang, and Liusheng Huang. 2018. CountMax: A Lightweight and Cooperative Sketch Measurement for Software-Defined Networks. *IEEE/ACM Transactions on Networking (TON)* 26, 6 (2018), 2774–2786.

[45] Yutong Zhai, Hongli Xu, Haibo Wang, Zeyu Meng, and He Huang. 2020. Joint Routing and Sketch Configuration in Software-Defined Networking. *IEEE/ACM Transactions on Networking* 28, 5 (2020), 2092–2105.

[46] Yinda Zhang, Jinyang Li, Yutian Lei, Tong Yang, Zhetao Li, Gong Zhang, and Bin Cui. 2020. On-off sketch: A fast and accurate sketch on persistence. *Proceedings of the VLDB Endowment* 14, 2 (2020), 128–140.

[47] Bohan Zhao, Xiang Li, Boyu Tian, Zhiyu Mei, and Wenfei Wu. 2021. DHS: Adaptive Memory Layout Organization of Sketch Slots for Fast and Accurate Data Stream Processing. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2285–2293.

[48] Yikai Zhao, Zheng Zhong, Yuanpeng Li, Yi Zhou, Yifan Zhu, Li Chen, Yi Wang, and Tong Yang. 2021. Cluster-Reduce: Compressing Sketches for Distributed Data Streams. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2316–2326.

[49] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. 2018. Cold filter: A meta-framework for faster and more accurate stream processing. In *Proceedings of the 2018 International Conference on Management of Data*. 741–756.

[50] You Zhou, Youlin Zhang, Chaoyi Ma, Shigang Chen, and Olufemi O Odegbile. 2019. Generalized sketch families for network traffic measurement. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 3 (2019), 1–34.

A PSEUDO-CODE

Algorithm 1: Insertion of the MimoSketch algorithm.

```
Input: An item e_i
if fp(e_i) exists in B[h(e_i)] then
    B[h(e_i)].f[e_i].fq += s(e_i);
    if B[h(e_i)].f[e_i].fq overflows then
        // find exchangeable higher-level slots
        Exchange (B[h(e_i)].f[e_i]);
        if fails then
            // adaptively adjust slot structure
            LevelUp (B[h(e_i)].f[e_i]);
            if fails then
                // expel the slot to sharing counter
                B[h(e_i)].count += B[h(e_i)].f[e_i].fq;
                Set status of B[h(e_i)].f[e_i] false;
else if slots in B[h(e_i)] are not full then
    Insert it into an empty slot with <fp(e_i), s(e_i)>;
else
    B[h(e_i)].count += s(e_i);
return;
```

B MATHEMATICAL PROOF AND ANALYSIS

In this section, we prove and analyze the mathematical properties of MimoSketch listed in Section 4. The symbols which may be frequently used in the mathematical analysis are shown in Table 5.

B.1 Proof of Unbiasedness

We prove the unbiasedness of MimoSketch algorithm as below.

Let Ω denote the set of all functions $s : \mathsf{E} \to \{+1, -1\}$, and $s(.)$ is a random function from Ω. Then, $s(e_i)$ and $s(e_j)(i < j)$ are two independent variables.

(1) An item is counted either in a slot or in the sharing counter, and would not be counted at two places (proven by the discussion in Section 3.2.2).

(2) If the item e_i is eventually in a slot, then $\mathsf{E} = \mathsf{E}_i$, and $\hat{f}_i = f_i^{\mathsf{E}_i}$,

$$\begin{aligned} E(\hat{f}_i) &= E\left(f_i^{\mathsf{E}_i}\right) = E\left(\sum_{e_j \in \mathsf{E}_i} f_j \cdot s(e_j) \cdot s(e_i)\right) \\ &= E\left(\sum_{e_j \in \mathsf{E}_i, j<i} f_j \cdot s(e_j) \cdot s(e_i)\right) + E\left(\sum_{j=i} f_j \cdot s(e_j) \cdot s(e_i)\right) \\ &= 0 + f_i = f_i. \end{aligned}$$

(3) If the item e_i is eventually in the sharing counter, it is stored together with fingerprint-collided items E_i and non-collided items $\overline{\mathsf{E}_i}$. Thus, $\hat{f}_i = f_i^{\mathsf{E}_i} + f_i^{\overline{\mathsf{E}_i}}$, and

$$\begin{aligned} E(\hat{f}_i) &= E\left(f_i^{\mathsf{E}_i} + f_i^{\overline{\mathsf{E}_i}}\right) \\ &= E\left(f_i^{\mathsf{E}_i}\right) + E\left(\sum_{e_j \in \overline{\mathsf{E}_i}} f_j \cdot s(e_j) \cdot s(e_i)\right) \qquad (7) \\ &= f_i + 0 = f_i. \end{aligned}$$

The second term in Equation (7) is zero because e_i is different from $\forall e_j \in \overline{\mathsf{E}_i}$.

We conclude that the expected value equals the actual frequency (*i.e.*, $E(\hat{f}_i) = f_i$) under any circumstance. Therefore, MimoSketch achieves an unbiased estimation.

B.2 Analysis of Variance and Error Bound

Variance. *Let $e_1, e_2, \ldots, e_n$ be the items hashed into $B[h(e_i)]$, the bound of the variance of MimoSketch's estimation is that*

$$Var\left(\hat{f}_i\right) \le \sum_{e_j < e_i} (f_j)^2.$$

Proof. Whether e_i is stored in a slot with a fingerprint or sharing counter, we both have $\hat{f}_i = \left(\sum_{e_j \in \mathsf{E}} f_j \cdot s(e_j)\right) \cdot s(e_i)$. The variance of $\hat{f}_i$ is that

$$\begin{aligned} Var\left(\hat{f}_i\right) &= \underset{s(e_j)\in\{1,-1\}}{E}\left(\sum_{e_j \in \mathsf{E}} f_j \cdot s(e_j) \cdot s(e_i) - E\left(\hat{f}_i\right)\right)^2 \\ &= \underset{s(e_j)\in\{1,-1\}}{E}\left(\sum_{e_j \in \mathsf{E} \wedge j<i} f_j \cdot s(e_j) \cdot s(e_i)\right)^2 \qquad (8) \\ &= \underset{s(e_j)\in\{1,-1\}}{E}\left(\sum_{e_j \in \mathsf{E} \wedge j<i} f_j \cdot s(e_j)\right)^2 \\ &= \sum_{e_j \in \mathsf{E} \wedge j<i} \underset{s(e_j)\in\{1,-1\}}{E}\left(f_j \cdot s(e_j)\right)^2 \qquad (9) \\ &\le \sum_{e_j < e_i} (f_j)^2. \end{aligned}$$

The Equation (8) holds because $E(\hat{f}_i) = f_i$ according to Section 4.1, and thus $f_i \cdot s(e_i)^2$ can eliminate $E(\hat{f}_i)$. The Equation (9) stands because different $f_j \cdot s(e_j)$ are independent, and thus the expectation of the cross term $s(e_i) \cdot s(e_j)(i < j)$ is zero.

With the bound of the variance, we can derive the upper and lower error bound.

Error bound based on $\|f\|_2$. *Let* $l = \frac{e}{\epsilon^2}$, $\|f\|_2 = \sqrt{\sum_{e_i \in \mathcal{S}} (f_i)^2}$,

$$P\left(\hat{f}_i - f_i \ge \epsilon \|f\|_2\right) \le \frac{1}{e}.$$

Proof. Due to Chebyshev's inequality,

$$P\left(\hat{f}_i - f_i \ge \sqrt{e \sum_{e_j < e_i} (f_j)^2}\right) \le \frac{Var\left(\hat{f}_i\right)}{e \sum_{e_j < e_i} (f_j)^2} \le \frac{1}{e}.$$

Table 5: List of Symbols Used for Mathematical Analysis

Notation	Meaning
f_i	the actual frequency of e_i
$\hat{f}_i$	estimated frequency of e_i from MimoSketch
E	$\mathsf{E} = \{e_1, e_2, \ldots, e_{n_i}\}$, containing all the distinct items stored in the same place as e_i
E_i	$\mathsf{E}_i = \{e_j \in \mathsf{E} \mid fp(e_j) = fp(e_i)\}$, containing the items whose fingerprint is the same as e_i's
$\overline{\mathsf{E}_i}$	$\overline{\mathsf{E}_i} = \mathsf{E} \setminus \mathsf{E}_i$, containing the remaining items
$f_i^{\mathsf{E}_i}$	$f_i^{\mathsf{E}_i} = \sum_{e_j \in \mathsf{E}_i} f_j \cdot s(e_j) \cdot s(e_i)$
$f_i^{\overline{\mathsf{E}_i}}$	$f_i^{\overline{\mathsf{E}_i}} = \sum_{e_j \in \overline{\mathsf{E}_i}} f_j \cdot s(e_j) \cdot s(e_i)$
e	total number of distinct items

Moreover, considering items in $B[h(e_i)]$, we can have an estimation that $\sum_{e_j \in S_i} (f_j)^2 = \frac{1}{l}(\|f\|_2)^2$, thus

$$\begin{aligned} P\left(\hat{f}_i - f_i \geq \epsilon \|f\|_2\right) &\leq P\left(\hat{f}_i - f_i \geq \epsilon \sqrt{\frac{\sum_{h(e_j)=h(e_i)} (f_j)^2}{l}}\right) \\ &\leq P\left(\hat{f}_i - f_i \geq \sqrt{e \sum_{e_j < e_i} (f_j)^2}\right) \\ &\leq \frac{1}{e}. \end{aligned}$$

Also, we can derive an error bound based on $\|f\|_1$ (*i.e.*, $\sum_{e_i \in \mathcal{S}} |f_i|$).

Error bound based on $\|f\|_1$. *Let* $l = \frac{e}{\epsilon}$, $\|f\|_1 = \sum_{e_i \in \mathcal{S}} |f_i|$,

$$P\left(\hat{f}_i - f_i \geq \epsilon \|f\|_1\right) \leq \frac{1}{e}.$$

Proof. We have

$$E\left(\hat{f}_i - f_i\right) = E\left(\sum_{e_j < e_i} f_j \cdot s(e_j)\right) \leq E\left(\sum_{e_j < e_i} f_j\right) \leq \frac{\epsilon}{e} \|f\|_1 .$$

By Markov's inequality,

$$P\left(\hat{f}_i - f_i \geq \epsilon \|f\|_1\right) \leq P\left(\hat{f}_i - f_i \geq eE\left(\hat{f}_i - f_i\right)\right) \leq \frac{1}{e}.$$

C DETAILS ABOUT EXPERIMENT SETUP

We list the details of our experiments for reproducibility.

Implementation. All the five algorithms are implemented in C++ and open-sourced at GitHub [2]. We deploy the previous work based on the source codes by their authors and use best-tuned parameters with the best performance for fairness. For Count Sketch, the hash number is set to 3 as a trade-off between accuracy and speed, and its query method is median-query which outperforms mean-query according to [7]. For DHS, we set bucket $W = 64$ bits and decay parameter $b = 1.08$, which performs the best results as concluded in [47]. For Cuckoo Counter, we set its *maxloop* = 2 with better accuracy. For MimoSketch, $d = 8$, $k = 3$, the *fingerprint* field is 8 bits, and the *frequency* field is 8, 12, and 16 bits, respectively. We use efficient hash functions MurmurHash [1] and BKDRHash [23] in each algorithm. The MimoSketch's policy is a MIP model, and we implement the policy based on Gurobi 9.5.2 optimization solver (Linux64, C++).

Test platform. We perform all the experiments on a machine with 2 Intel i7-11700K CPUs, each with 8 cores @ 3.60GHz, and 32 GB DRAM memory. In order to smooth the CPU jitter errors, we conduct each experiment ten times and take the average.

Datasets. We use two real datasets for experiments. One is CAIDA Internet Trace [5], collected by Equinix-Chicago monitor from CAIDA. The item inside data streams of the traces are IP packets identified by 5-tuple headers (*i.e.*, Source IP, Destination IP, Source Port, Destination Port, and Protocol). The dataset contains 1.6M unique items with 25M occurrences in total. The other is WebDocs Dataset, which is a real-life huge transactional dataset built from a collection of web documents, downloaded from its website [13]. There are 1M distinct items with around 32M occurrences in total. More details about the dataset can be obtained at its website [8].

Baselines. We compare our MimoSketch (MIMO) with four related algorithms. Among them, Count Sketch (CS) [7], WavingSketch (WS) [25] are also unbiased algorithms; Dynamic Hierarchical Sketch (DHS) [47], Cuckoo Counter (CC) [32] are biased ones. We choose these algorithms because they have similar properties or mechanisms to MimoSketch and show comparable or better performance than other prior works.

Task. The main mining task of item frequency estimation is formulated as follows. It offers the approximately estimated frequency of each item, *i.e.*, $\hat{f}_1, \ldots, \hat{f}_n$. As the fundamental estimation task for data stream mining, it is implemented by querying each item and comparing the estimated size to the actual frequency, using AAE and ARE as main metrics (defined as below).

Metrics. We choose the following metrics to evaluate the item frequency estimation task.

Average Absolute Error (AAE): $\frac{1}{|\Psi|} \sum_{e_i \in \Psi} |f_i - \hat{f}_i|$, where Ψ is the query set, f_i is the actual frequency of item e_i, and $\hat{f}_i$ is the estimated frequency of e_i.

Average Relative Error (ARE): $\frac{1}{|\Psi|} \sum_{e_i \in \Psi} |f_i - \hat{f}_i|/|f_i|$, where the parameters are the same as those in AAE.

Throughput: N_I/T, normally in the unit of Mps, where N_I is the number of insertion operations, and T is the total time. Different from the aforementioned metrics evaluating the accuracy, throughput is used to measure the processing speed.

MIMO prototype. In the MIMO-scenario evaluation in Section 6.3, we consider two datasets as data streams, and assume there are seven nodes responsible for them. For each item, the subset of candidate node(s) is randomly chosen and the cardinality of the set is 1, 3, or 5, as described in Figure 4.