Near-Optimal Per-Key Streaming Quantile Estimation

JIARUI GUO, Peking University, China
FEIYU WANG, Peking University, China
ZHUOCHEN FAN*, Pengcheng Laboratory, China
TONG YANG*, Peking University, China
XIAOLIN WANG, Peking University, China

Quantile estimation plays a pivotal role in the streaming computational model. In 2016, Karnin, Lang, and Liberty proposed the KLL sketch, which is widely recognized as the optimal algorithm for randomized streaming quantile estimation. Recently, per-key quantile estimation has attracted considerable attention from researchers. In such scenarios, every item in the data stream represents a key-value pair (k, v), and the goal is to estimate quantiles for all heavy hitters — keys with a proportion exceeding a certain threshold θ in the data stream. Per-key quantile estimation has wide applications in network measurement, data management, and anomaly detection, and several solutions have been proposed to address this problem. However, while these approaches have demonstrated empirical effectiveness, they either lack rigorous error guarantees or incur suboptimal space complexity, leaving a gap between theory and practice.

In this paper, we propose KLL-Polymer, the first per-key quantile sketch to achieve near-optimal space complexity. KLL-Polymer inherits the idea of hierarchical sampling from the KLL sketch, but aggregates items by key during the sampling process, thus enabling a single sketch to efficiently handle multiple keys, rather than maintaining separate KLL sketches for each key. Consequently, KLL-Polymer achieves a space complexity of $O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$ while ensuring that the probability of the error exceeding ε is at most δ, which is very close to the theoretical lower bound of $O\left(\frac{1}{\theta\varepsilon}\log\log\frac{1}{\delta}\right)$. We further introduce a deterministic-compaction variant of KLL-Polymer, which optimizes its performance by more effectively filtering infrequent keys at the top levels. The experimental results demonstrate that, with minimal additional overhead, KLL-Polymer significantly reduces estimation error compared to existing methods. Furthermore, we integrate KLL-Polymer into the RocksDB database, where it successfully accelerates distribution monitoring and reduces tail latency for per-key quantile queries.

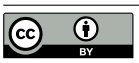
CCS Concepts: • **Theory of computation → Sketching and sampling**; *Streaming models*; *Data structures design and analysis.*

Additional Key Words and Phrases: Quantiles, Sketches, Streaming algorithms, KLL sketch

ACM Reference Format:
Jiarui Guo, Feiyu Wang, Zhuochen Fan, Tong Yang, and Xiaolin Wang. 2026. Near-Optimal Per-Key Streaming Quantile Estimation. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 280 (September 2026), 30 pages. https://doi.org/10.1145/3837118

*Zhuochen Fan and Tong Yang are corresponding authors.

Authors' Contact Information: Jiarui Guo, State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China, ntguojiarui@pku.edu.cn; Feiyu Wang, State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China, wangfeiyu@pku.edu.cn; Zhuochen Fan, Department of Strategic and Advanced Interdisciplinary Research, Pengcheng Laboratory, Shenzhen, China, fanzc@pku.edu.cn; Tong Yang, State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China, yangtong@pku.edu.cn; Xiaolin Wang, State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China, wxl@pku.edu.cn.



ACM 2836-6573/2026/9-ART280
https://doi.org/10.1145/3837118

1 Introduction

Quantile is a fundamental notion in statistics and data analysis [1–4]. For a multiset of N items $S = \{v_1, \cdots, v_N\}$, the quantile of v is the proportion of items that are less than or equal to v. Equivalently, define $\text{rank}(v)$ as the number of items with $v_i \leq v$, then the quantile of v is simply $\frac{\text{rank}(v)}{N}$. Given two small positive numbers ε and δ, the goal of quantile estimation is to maintain an estimate $\widehat{\text{rank}}(v)$, *s.t.* the estimation error $\left|\widehat{\text{rank}}(v) - \text{rank}(v)\right|$ is within εN, either deterministically or randomly (*i.e.* with probability at least $1 - \delta$).

Quantiles arise in a wide range of tasks [5–9]. In certain scenarios, data arrives continuously at high volume [10–12], making it impractical to store or revisit the entire dataset. This motivates the study of streaming quantile estimation, where the task is to maintain high-accuracy quantile summaries on-the-fly [13–15]. Unlike classical offline algorithms, a streaming algorithm must process each item in a *single pass* — once an item arrives, it must be handled immediately, and cannot be revisited [16–18]. Moreover, the algorithm may retain only a compact summary of the data stream, typically using $o(N)$ space. This ensures that the memory consumption remains sublinear even when processing massive, potentially unbounded data streams.

In streaming settings, the **KLL sketch** [19] stands as the asymptotically optimal algorithm for randomized quantile estimation. Its core idea is hierarchical sampling: it maintains a sketch of H levels, each storing a small subset of items sampled from the stream, up to a fixed per-level capacity. New items enter the sketch at the lowest level, and once the number of items reaches its capacity, the KLL sketch *compacts* it by discarding half of the items and promoting the remaining half of the items to the next level, forming a multi-level summary for quantile estimation. In addition, the KLL sketch introduces several refinements to reduce space usage further: ① replacing the lowest levels with an $O(1)$-space sampler; ② fixing the capacity of the highest levels to a constant; and ③ incorporating GK sketches — the optimal deterministic quantile sketch [20] at the top levels. Overall, the KLL sketch achieves a space complexity of $O\left(\frac{1}{\varepsilon}\log\log\frac{1}{\delta}\right)$.

However, real-world data stream analysis is often more complex. In many practical scenarios, data arrives as key-value pairs [21, 22], and it is often necessary to estimate quantiles separately for each key. Moreover, in some applications, multiple sub-streams — each corresponding to a different key — are merged into a single stream [23, 24], and such estimation must be performed individually for each key in the combined stream. These give rise to the **per-key quantile estimation** problem, where the target is to maintain quantile summaries for each key individually in the streaming setting.[1] In practice, data distributions are often highly skewed, with a small number of keys appearing frequently [25, 26]; hence, the primary goal of per-key quantile estimation is to provide accurate estimates for these heavy hitters, *i.e.* keys with a proportion exceeding a certain threshold θ.

Building on the focus on heavy hitters, per-key quantile estimation has numerous practical applications across diverse domains. For example, in network services, per-key quantile estimation enables monitoring of latency, throughput, and flow sizes at the granularity of individual flows, which helps identify the quality of service experienced by each client [27, 28]. In database management systems, per-key quantile summaries capture value distributions across different attributes and partitions, thereby supporting efficient selectivity estimation, query planning, and adaptive re-optimization [29, 30]. Furthermore, maintaining per-key quantiles allows the system to identify abnormal behavior within specific entities, *e.g.* individual users, services, or devices, which is crucial for detecting network failures and security threats [31, 32].

[1]The formal definition of per-key quantile estimation is shown in Section 2.

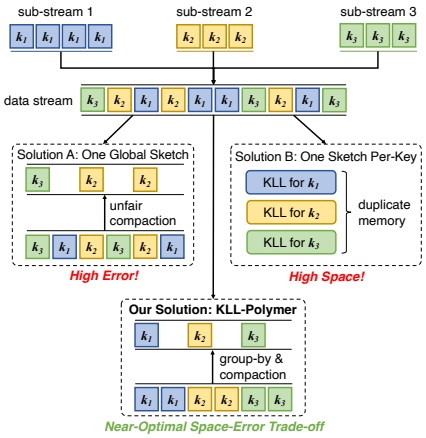


Fig. 1. Challenges of per-key extensions of KLL.

Several algorithms have been proposed to address the per-key quantile estimation problem, including HistSketch [33], SQUAD [34, 35], SketchPolymer [36], and M4 [37, 38]. However, in contrast to the single-key setting — where algorithms like the KLL sketch enjoy tight theoretical guarantees — most per-key solutions are primarily evaluated through empirical performance. Specifically, HistSketch, SketchPolymer, and M4 rely on strong assumptions about the underlying data distribution, and thus lack universal and rigorous space complexity bounds in general streaming settings. Although SQUAD establishes a formal space bound of $O\left(\frac{1}{\theta\varepsilon^{1.5}}\log\frac{1}{\delta}\right)$, its $\widetilde{O}\left(\varepsilon^{-1.5}\right)$ dependency on the error parameter ε remains asymptotically suboptimal compared to the linear $\widetilde{O}\left(\varepsilon^{-1}\right)$ dependency of the KLL sketch, leaving considerable room for further improvement.

This theoretical gap naturally motivates us to revisit the KLL sketch, the optimal solution in single-key settings. Ideally, one would seek to apply the KLL sketch to the per-key setting. However, achieving such an extension is non-trivial. As illustrated in Figure 1, since values belonging to different keys are not comparable, maintaining one global KLL sketch and still applying the standard compaction mechanism can induce high error, as it risks disproportionately discarding items from certain keys, thereby distorting their rank estimates. Alternatively, allocating one KLL sketch per key is space-inefficient, as the number of keys can far exceed the number of heavy hitters, inevitably wasting memory on infrequent keys. *This space-error conflict necessitates a novel sketching approach that achieves the accuracy of per-key isolation while maintaining the space efficiency of a global structure.*

In this paper, we propose **KLL-Polymer**, a near-optimal randomized sketching algorithm for per-key quantile estimation. KLL-Polymer augments each compaction operation of the KLL sketch with a lightweight *group-by* step, ensuring that items are compacted within each key. This simple yet crucial modification preserves the essential properties of the KLL sketch while enabling per-key estimation. Furthermore, KLL-Polymer retains the original optimizations ① and ②, yielding a mergeable per-key data structure with a total space complexity of $O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$, which is remarkably close to the theoretical lower bound of $O\left(\frac{1}{\theta\varepsilon}\log\log\frac{1}{\delta}\right)$. We also propose a *deterministic-compaction* variant of KLL-Polymer (**KLL-Polymer-DC** for short). This variant sacrifices mergeability for more efficient filtering of infrequent keys at the top levels, potentially achieving superior performance on real-world data streams. Our experiments demonstrate that KLL-Polymer reduces quantile estimation error by up to 29.58× compared to the state-of-the-art algorithm SQUAD. Moreover, KLL-Polymer achieves higher insertion throughput than SQUAD, making it highly competitive for large-scale data streams. All related code is available on GitHub [39].

The remainder of this paper is organized as follows: Section 2 formalizes the per-key quantile estimation problem and reviews prior work on quantile sketches. Section 3 presents the basic version of KLL-Polymer and establishes its space complexity. Section 4 introduces two key optimizations of KLL-Polymer and develops the KLL-Polymer-DC variant; it also provides the space lower bound for per-key quantile estimation and compares KLL-Polymer with existing per-key sketches. Section 5 conducts an extensive experimental evaluation, demonstrating the superiority of KLL-Polymer over prior work and detailing its practical integration into the RocksDB database. Finally, Section 6 summarizes our contributions and concludes the paper.

2 Problem Definition and Related Work

2.1 Problem Definition

Table 1 summarizes the symbols used in the problem definition.

Notation	Meaning
S	The data stream
N	Number of items in the data stream
e	An item in the data stream
k	Key of a certain item
v	Value of a certain item
θ	Threshold of heavy hitters
ε	Error threshold of quantile estimation
δ	Failure probability of quantile estimation
f_k	Frequency of key k
$\hat{f}_k$	Estimated frequency of key k
$\mathrm{rank}(k, v)$	Rank of v among values with key k
$\widehat{\mathrm{rank}}(k, v)$	Estimated rank of v among values with key k

Table 1. Symbols frequently used in the problem definition.

We formalize the per-key streaming quantile estimation problem as follows.

Definition 1. *A **data stream** $S = \{e_1, e_2, \cdots, e_N\}$ is a series of N items arriving over time. In this paper, each item e_i in the data stream is a key-value pair (k_i, v_i), where k_i denotes its key, and v_i denotes its value.*

Definition 2. *For an arbitrary key k, its **frequency** is defined as*

$$f_k = \sum_{e_i \in S} \mathbb{I}_{k_i = k},$$

*where the indicator variable $\mathbb{I}_A = 1$ if A is true, and $\mathbb{I}_A = 0$ if A is false. Given a threshold $\theta \in (0, 1)$, keys with $f_k \geq \theta N$ are referred to as **heavy hitters**.*

Definition 3. *Let $\theta, \varepsilon, \delta \in (0, 1)$. For a heavy hitter k, let $\mathrm{rank}(k, v)$ denote the rank of v among all values associated with key k in the data stream. The goal of the **per-key quantile estimation** problem is to construct a data structure that maintains an estimate $\widehat{\mathrm{rank}}(k, v)$, such that*

$$\left| \widehat{\mathrm{rank}}(k, v) - \mathrm{rank}(k, v) \right| \leq \varepsilon f_k$$

with probability at least $1 - \delta$.

2.2 Single-Key Quantile Sketches

The problem of single-key quantile estimation, first introduced by Munro and Paterson in 1980 [40], has been the primary focus of research in this area. In the single-key setting, each item in the data stream is a value, and the goal is to estimate the rank of an arbitrary value v, such that $\left|\widehat{\text{rank}}(v) - \text{rank}(v)\right| \leq \varepsilon N$, where $\widehat{\text{rank}}(v)$ and $\text{rank}(v)$ denote the estimated and true ranks of v, respectively. Most works assume the *comparison model*, where a total order is imposed on the values, and the algorithm can only compare two values at a time to determine which is larger. In 1999, Manku, Rajagopalan, and Lindsay proposed an algorithm using $O\left(\frac{1}{\varepsilon}\log^2(\varepsilon N)\right)$ space [41]. Subsequently, Greenwald and Khanna designed a more sophisticated algorithm with a space complexity of $O\left(\frac{1}{\varepsilon}\log(\varepsilon N)\right)$ [20], which was later proved to be optimal among deterministic algorithms [42].

Later studies introduced randomization to provide probabilistic guarantees, ensuring that $\left|\widehat{\text{rank}}(v) - \text{rank}(v)\right| \leq \varepsilon N$ holds with probability at least $1 - \delta$. The incorporation of randomization enables sampling-based strategies, and Felber and Ostrovsky achieved a space complexity of $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ by sampling items from the data stream and inserting them into a GK sketch [43]. Finally, Karnin, Lang, and Liberty proposed the KLL sketch [19], which achieves the optimal space complexity of $O\left(\frac{1}{\varepsilon}\log\log\frac{1}{\delta}\right)$ for streaming quantile estimation.

Beyond the comparison model, single-key quantile estimation has also been studied under additional assumptions about the data stream. Guha and McGregor [44] proposed an algorithm with a space complexity of $O\left(\frac{1}{\varepsilon}\right)$ for $\varepsilon = O\left(\frac{1}{\sqrt{N}}\text{poly}\log\frac{N}{\delta}\right)$ when items in the data stream arrive in random order. Other algorithms observed that the data stream is often sparse in reality, and accordingly designed methods to improve practical performance, *e.g.* DDSketch [45], t-digest [46], *etc.* However, due to their assumptions about the data distribution, these algorithms usually lack space complexity guarantees and no longer operate under the comparison model, and thus are not considered in this paper.

2.3 Per-Key Quantile Sketches

Several algorithms have been proposed to directly estimate per-key quantiles. HistSketch [33] partitions all values into multiple intervals and employs a two-stage data structure to maintain information for different keys. Heavy hitters are stored in the first stage with full information, while keys with lower frequencies are handled in the second stage, sharing counters to reduce memory usage. SketchPolymer [36] similarly partitions values into logarithmic intervals, and encodes key-interval information using Count-Min Sketches [47] and Bloom Filters [48] to improve both storage and computational efficiency. M4 [37, 38] maps the values of heavy hitters into buckets at different levels. Within each bucket, M4 applies a single-key algorithm META (*e.g.* DDSketch [45], t-digest [46], ReqSketch [49, 50], *etc.*) to estimate the distribution of all mapped values. Finally, the results from different buckets are merged to obtain a per-key distribution.

However, all of these algorithms discretize values into multiple intervals, which implicitly relies on assumptions about the value distribution. Consequently, they are not based on the comparison model and fail to provide rigorous error bounds under this setting. Moreover, they employ different techniques to filter infrequent keys in advance, which potentially increases the assumptions made about the data stream distribution. In contrast, comparison-based algorithms offer practical advantages by overcoming these inherent limitations: Specifically, they are domain-parameter-free, requiring no prior knowledge of value ranges or universe sizes [42]. Furthermore, they exhibit remarkable robustness in dense-interval scenarios, maintaining high accuracy even when data is

highly concentrated within a narrow range, a situation where interval-based methods severely degrade [50].

To the best of our knowledge, SQUAD [34, 35] is the only algorithm that provides a universal space complexity guarantee under the comparison model. It employs the Space Saving algorithm [51] to maintain heavy hitter information in sketches, and uses Reservoir Sampling [52] to capture the values that arrive before a heavy hitter is successfully inserted. In this way, SQUAD achieves a space complexity of $O\left(\frac{1}{\theta\varepsilon^{1.5}}\log\frac{1}{\delta}\right)$.

2.4 The KLL Sketch

Since the KLL sketch serves as the optimal solution for single-key streaming quantile estimation, and it forms the foundation of KLL-Polymer in the per-key setting, we introduce its data structure and operations in detail in this section.

The KLL sketch [19] is a hierarchical sampling-based data structure that performs compaction operations to maintain approximate quantile summaries. It is organized into H levels: level h has capacity $c_h = \max\{\lceil l \cdot c^{H-h}\rceil, 2\}$, for $h = 1, 2, \cdots, H$, where $l > 0$ and $\frac{1}{2} < c < 1$ are constants. Each item in level h carries a weight of 2^{h-1}. In the single-key scenario, each item of the data stream is simply a value v_i. When a new item arrives, it is initially placed in level 1. Once the number of items in level h reaches its capacity, the KLL sketch sorts all items in this level according to their values, randomly keeps either the items at odd positions or at even positions, and promotes them to level $h+1$. To query the rank of v, the KLL sketch selects all items less than or equal to v and sums their weights to obtain the estimated rank of v. In this way, the KLL sketch guarantees $\left|\widehat{\text{rank}}(v) - \text{rank}(v)\right| \le \varepsilon N$ with probability at least $1-\delta$ using $O\left(\frac{1}{\varepsilon}\sqrt{\log\frac{1}{\delta}} + \log(\varepsilon N)\right)$ space.

The KLL sketch further introduces several optimizations to reduce space complexity. First, the lowest levels of the KLL sketch with a capacity of 2 are compressed into a bucket using Reservoir Sampling [52], which brings the space complexity down to $O\left(\frac{1}{\varepsilon}\sqrt{\log\frac{1}{\delta}}\right)$. Next, the capacities of the highest few levels of the KLL sketch are fixed to a constant instead of exponentially decreasing, yielding a space cost of $O\left(\frac{1}{\varepsilon}\log^2\log\frac{1}{\delta}\right)$ while maintaining mergeability. Finally, by sacrificing mergeability, the top level of the KLL sketch can be replaced with the GK sketch [20], achieving $O\left(\frac{1}{\varepsilon}\log\log\frac{1}{\delta}\right)$ space complexity, which is asymptotically optimal in the single-key quantile estimation problem.

Due to the simplicity and optimality of the KLL sketch, many variants have been proposed subsequently. In 2021, Zhao *et al.* proposed KLL$^{\pm}$ [53], which extends the KLL sketch to support the deletion operation. In 2024, Chen *et al.* designed randomized sketches based on the KLL sketch for quantile estimation in LSM-based databases [54]. More recently, Shi *et al.* observed the sparsity of real-world data streams and proposed Cooled-KLL [55] to improve the practical performance of the KLL sketch.

3 KLL-Polymer Algorithm

In this section, we introduce the data structure and operations of KLL-Polymer, followed by an analysis of its space complexity. Further optimizations of KLL-Polymer are discussed in the next section. Table 2 summarizes the symbols used for KLL-Polymer in Sections 3 and 4.

Notation	Meaning
H	Number of levels
h	An arbitrary level
c_h	Capacity of level h
w_h	Weight of items of level h
l	Capacity of the highest level (*i.e.* c_H)
c	Capacity ratio of two adjacent levels (*i.e.* $\frac{c_{h+1}}{c_h}$)
L	Number of bottom levels replaced by a sampler
s	Number of top levels with fixed capacity
C, C_1, C_2	Certain large constants

Table 2. Symbols of KLL-Polymer.

3.1 Data Structure and Operation

Data Structure: KLL-Polymer consists of H levels, where level h has a capacity of $c_h = \max\left\{\lceil l \cdot c^{H-h}\rceil, 2\right\}$, with constants $l > 0$ and $\frac{1}{2} < c < 1$. Unlike the standard KLL sketch, each item in level h is a key–value pair (k, v), representing a sampled item from the data stream. Items in level h carry a weight of 2^{h-1}. All items in the data stream are first inserted into level 1. When the number of items in level h reaches its capacity, a compaction operation is triggered, forwarding half of the items into level $h+1$.

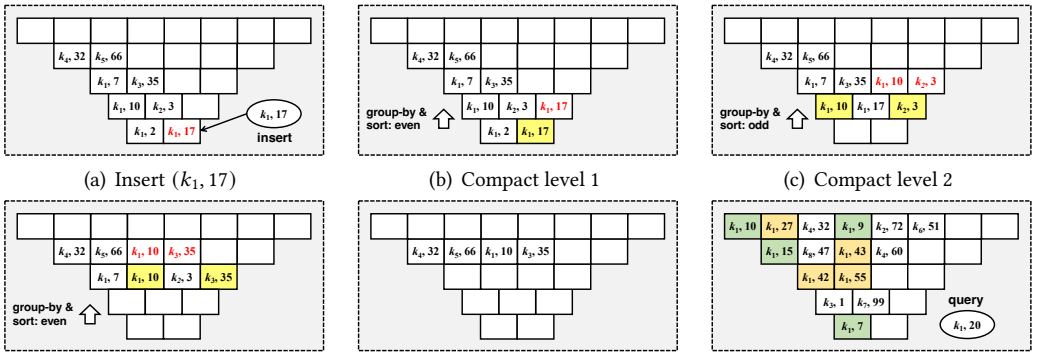
Key Operation: Compaction. Similar to the KLL sketch, KLL-Polymer has to compact items in each level once it is full. The compaction operation works as follows: To compact c_h items in level h into level $h+1$, KLL-Polymer performs a **group-by** operation on items based on their keys, *i.e.* items with the same key are put in the same group. KLL-Polymer then sorts values in each group and randomly selects either the odd- or even-indexed items to the upper level.[2] Finally, all stored items in level h are cleared to make room for later items in the data stream.

Explanation: Why Group-by Plus Sorting Rather Than Direct Sorting? KLL-Polymer applies a group-by operation before sorting for two main reasons: ① Values associated with different keys are not comparable; ② Directly sorting all items may cause the values of a certain key to be unevenly distributed between odd and even positions, leading to large changes in its rank after compaction. For example, if key k_1 receives odd values $(1, 3, \cdots, 99)$ and key k_2 receives even values $(2, 4, \cdots, 100)$, a direct global sort places k_1 entirely at odd indices and k_2 at even indices. Consequently, a compaction (*i.e.*, evicting either all odd or even positions) would cause a 100% information loss for either k_1 or k_2. By introducing a group-by operation before sorting, KLL-Polymer ensures that the values of each key are compacted in the same manner as in a standard KLL sketch, thereby limiting the potential change of $\text{rank}(k, v)$ to at most 2^{h-1} at level h.

Implementation of Compaction Operation: To achieve the group-by plus sorting operation within compaction, we implement a highly efficient hash-based approach. Specifically, we allocate a vector for each distinct key, distribute the values into their corresponding vectors, sort the values within each vector, and finally concatenate them for the subsequent odd/even selection. This method ensures efficiency with an auxiliary space overhead of $O(c_h)$, where c_h is the capacity of the compacted level. Notably, we strictly avoid the naive mechanism of first sorting by key, then sorting by value, as KLL-Polymer does not assume a natural order among keys.[3]

[2]To simplify the compaction operation, KLL-Polymer just rearranges items in level h into $\{e_1, \cdots, e_{n_1}\}$, $\{e_{n_1+1}, \cdots, e_{n_1+n_2}\}$, $\cdots$, $\{e_{n_1+\cdots+n_{l-1}+1}, \cdots, e_{n_1+\cdots+n_l}\}$, while items within the same group have the same key and their values are ordered in non-decreasing order. Then a single random choice $j \in \{0, 1\}$ determines whether the odd- or even-indexed items in the entire sequence are forwarded to the upper level.

[3]Further details will be discussed later in Section 4.3.



(a) Insert $(k_1, 17)$ (b) Compact level 1 (c) Compact level 2

(d) Compact level 3 (e) Finish inserting (f) Query $(k_1, 20)$

Fig. 2. A Running Example of KLL-Polymer.

Query: To query $\text{rank}(k, v)$ for a given key k and value v, KLL-Polymer traverses all levels, identifies items with key k and value smaller than or equal to v, and sums their weights up as $\widehat{\text{rank}}(k, v)$. KLL-Polymer also supports frequency estimation: KLL-Polymer just locates all items associated with key k, and aggregates their weights to obtain its estimated frequency $\hat{f}_k$. Subsequently, KLL-Polymer computes the estimated quantile for key k and value v by $\hat{q}(k, v) = \frac{\widehat{\text{rank}}(k,v)}{\hat{f}_k}$. In addition, KLL-Polymer supports quantile queries: for a given key k and a quantile level $\phi \in (0, 1)$, KLL-Polymer returns the smallest value v whose estimated rank satisfies $\widehat{\text{rank}}(k, v) \geq \phi \hat{f}_k$.

3.2 A Running Example

Assume $l = 8$, $H = 5$ and $c = \frac{1}{\sqrt{2}}$. The running example of KLL-Polymer is shown in Figure 2.

Insertion: The initial state of KLL-Polymer is shown in Figure 2(a). To insert a new item $e = (k_1, 17)$, KLL-Polymer just inserts it into level 1. Since level 1 now reaches its capacity, we compact this level in Figure 2(b): KLL-Polymer groups items by key, orders them as $\{(k_1, 2), (k_1, 17)\}$, and randomly chooses either the odd- or even-indexed items to promote. Assuming the even-indexed items are selected, then $(k_1, 17)$ is promoted to level 2. After the compaction, level 2 becomes full with 3 items. Therefore, KLL-Polymer compacts level 2 by rearranging items into $\{(k_1, 10), (k_1, 17)\}$, $\{(k_2, 3)\}$, and promoting odd-indexed items, namely $(k_1, 10)$ and $(k_2, 3)$ to level 3 in Figure 2(c). Level 3, after receiving 2 items from level 2, also reaches its capacity of 4. KLL-Polymer again performs group-by and sorting in Figure 2(d), obtaining $\{(k_1, 7), (k_1, 10)\}$, $\{(k_2, 3)\}$, $\{(k_3, 35)\}$. Supposing items with even indexes are chosen this time, then $(k_1, 10)$ and $(k_3, 35)$ are fed into level 4. Since level 4 remains below capacity, the insertion procedure terminates. The final state of KLL-Polymer appears in Figure 2(e).

Query: The query process is shown in Figure 2(f), which reflects the state of the sketch at the end of the data stream after all items have been inserted. To query the rank for key k_1 and value 20, KLL-Polymer traverses all levels for items with key k_1 and values smaller than or equal to 20. Levels 1, 2, 3, 4 and 5 have 1, 0, 0, 1 and 2 such items, and the weights of items in these levels are 1, 2, 4, 8 and 16 respectively. Thus, the estimated rank of $(k_1, 20)$ is calculated as

$$\widehat{\text{rank}}(k_1, 20) = 1 \times 1 + 0 \times 2 + 0 \times 4 + 1 \times 8 + 2 \times 16 = 41.$$

Since levels 1, 2, 3, 4, and 5 store 1, 0, 2, 2, and 3 items with key k_1, the estimated frequency of k_1 is

$$\hat{f}_1 = 1 \times 1 + 0 \times 2 + 2 \times 4 + 2 \times 8 + 3 \times 16 = 73.$$

Finally, the quantile of $(k_1, 20)$ can be approximated as $\hat{q}(k_1, 20) = \frac{\widehat{\text{rank}}(k_1,20)}{\hat{f}_1} = \frac{41}{73} \approx 0.56$.

3.3 Space Complexity

In this part, we derive the space complexity of the basic KLL-Polymer algorithm step by step. We begin by introducing Hoeffding's Inequality as a lemma.

Lemma 1. ***Hoeffding's Inequality.*** *Let $X_1, \cdots, X_m$ be independent random variables, with $\mathbb{E}X_i = 0$ and $-w_i \le X_i \le w_i$. Then for any $t > 0$ we have*

$$\mathbb{P}\left(\left|\sum_{i=1}^{m} X_i\right| > t\right) \le 2\exp\left(-\frac{t^2}{2\sum_{i=1}^{m} w_i^2}\right).$$

Then, we show the space complexity of KLL-Polymer in Theorem 2.

Theorem 2. *KLL-Polymer achieves per-key quantile estimation with a space complexity of $O\left(\frac{1}{\theta\varepsilon}\sqrt{\log\frac{1}{\delta}} + \log(\theta\varepsilon N)\right)$.*

Proof. Let m_h denote the number of compaction operations at level h, where $h = 1, \cdots, H-1$. Since the capacity of level h is $c_h = \max\left\{\lceil l \cdot c^{H-h}\rceil, 2\right\}$, and items in this level carry a weight of $w_h = 2^{h-1}$, we have

$$m_h \le \frac{N}{c_h w_h} \le \frac{2N}{lc^H}\left(\frac{c}{2}\right)^h. \tag{1}$$

Specifically, we have $m_{H-1} \ge 1$ and $m_H \le 1$, hence

$$lc \cdot 2^{H-2} \le N \le l \cdot 2^{H-1}. \tag{2}$$

Now we bound the error of the compaction operation caused by the bottom H' levels ($1 \le H' \le H-1$). For the i^{th} compaction in level h, let $X_{i,h}$ denote the change of $\widehat{\text{rank}}(k, v)$ due to this compaction operation. Then, if the rank of v is even among all items in this level with key k, then its rank remains unchanged after compaction, *i.e.* $X_{i,h} = 0$; if the rank of v is odd, then its rank increases by w_h if odd items are chosen, and decreases by w_h if even items are chosen, *i.e.* $P(X_{i,h} = w_h) = P(X_{i,h} = -w_h) = \frac{1}{2}$. In both situations, $X_{i,h}$ is within $[-w_h, w_h]$ and its expectation is 0. Since

$$\sum_{h=1}^{H'}\sum_{i=1}^{m_h} w_h^2 = \sum_{h=1}^{H'} m_h w_h^2 \le \frac{N}{2lc^H}\sum_{h=1}^{H'}(2c)^h \le \frac{cN}{(2c-1)l\cdot c^H}(2c)^{H'},$$

by substituting Equation 2 into the above equation, we have

$$\begin{aligned}\sum_{h=1}^{H'}\sum_{i=1}^{m_h} w_h^2 &\le \frac{cN}{(2c-1)l\cdot c^H}(2c)^{H'}\cdot\frac{N}{lc\cdot 2^{H-2}}\\ &\le \frac{cN}{c^2(2c-1)l}(2c)^{H'}\cdot\frac{N}{lc\cdot(2c)^{H-2}}\\ &\le \frac{4N^2}{(2c-1)l^2}(2c)^{H'-H}.\end{aligned} \tag{3}$$

Set $H' = H-1$. Since

$$\widehat{\text{rank}}(k, v) - \text{rank}(k, v) = \sum_{h=1}^{H-1}\sum_{i=1}^{m_h} X_{i,h},$$

and $f_k \geq \theta N$, by applying Hoeffding's Inequality, we obtain

$$\begin{aligned}\mathbb{P}\left(\left|\sum_{h=1}^{H-1}\sum_{i=1}^{m_h} X_{i,h}\right| > \varepsilon f_k\right) &\leq 2\exp\left(-\frac{\varepsilon^2 f_k^2}{2\sum_{h=1}^{H-1}\sum_{i=1}^{m_h} w_h^2}\right)\\ &\leq 2\exp\left(-\frac{(2c-1)\varepsilon^2 f_k^2 l^2}{8N^2}\right) \leq 2\exp\left(-\frac{(2c-1)\theta^2\varepsilon^2 l^2}{8}\right).\end{aligned} \tag{4}$$

To ensure $\left|\widehat{\text{rank}}(k,v) - \text{rank}(k,v)\right| \leq \varepsilon f_k$ with probability at least $1-\delta$, we just set $l = \frac{2\sqrt{2}}{\theta\varepsilon\sqrt{2c-1}}\sqrt{\log\frac{2}{\delta}}$. The overall space complexity of KLL-Polymer is

$$\begin{aligned}\sum_{h=1}^{H} c_h &= \sum_{h=1}^{H} \max\left\{\lceil l\cdot c^{H-h}\rceil, 2\right\} \leq \sum_{h=1}^{H}\left(l\cdot c^{H-h}+3\right)\\ &\leq \frac{l}{1-c} + 3H = O\left(\frac{1}{\theta\varepsilon}\sqrt{\log\frac{1}{\delta}} + \log(\theta\varepsilon N)\right)\end{aligned}$$

by noticing that $H \leq 2 + \log\frac{N}{lc}$. □

Theorem 3. *For a heavy hitter k, KLL-Polymer also guarantees $\left|\hat{f}_k - f_k\right| \leq 2\varepsilon f_k$ with probability at least $1-2\delta$.*

Proof. Let $\text{RANK}(k,v)$ be the rank of v among values associated with key k, counted in descending order, and $\widehat{\text{RANK}}(k,v)$ be its estimated rank. Similar to Theorem 2, we have

$$\left|\widehat{\text{RANK}}(k,v) - \text{RANK}(k,v)\right| \leq \varepsilon f_k$$

with probability at least $1-\delta$. Since $f_k = \text{rank}(k,v) + \text{RANK}(k,v)$, applying the union bound we get

$$\begin{aligned}\left|\hat{f}_k - f_k\right| &= \left|\widehat{\text{rank}}(k,v) - \text{rank}(k,v) + \widehat{\text{RANK}}(k,v) - \text{RANK}(k,v)\right|\\ &\leq \left|\widehat{\text{rank}}(k,v) - \text{rank}(k,v)\right| + \left|\widehat{\text{RANK}}(k,v) - \text{RANK}(k,v)\right|\\ &\leq \varepsilon f_k + \varepsilon f_k = 2\varepsilon f_k\end{aligned}$$

with probability at least $1-2\delta$. □

Corollary 4. *For a heavy hitter k and an arbitrary value v, let $q(k,v) = \frac{\text{rank}(k,v)}{f_k}$ and $\hat{q}(k,v) = \frac{\widehat{\text{rank}}(k,v)}{\hat{f}_k}$ be its true and estimated quantile. Then KLL-Polymer guarantees $|\hat{q}(k,v) - q(k,v)| \leq \varepsilon$ with probability at least $1-2\delta$.*

Proof. Since

$$\begin{aligned}&|\hat{q}(k,v) - q(k,v)| = \left|\frac{\widehat{\text{rank}}(k,v)}{\hat{f}_k} - \frac{\text{rank}(k,v)}{f_k}\right|\\ =&\frac{1}{f_k\hat{f}_k}\cdot\left|\widehat{\text{rank}}(k,v)\cdot(\text{rank}(k,v)+\text{RANK}(k,v)) - \text{rank}(k,v)\cdot\left(\widehat{\text{rank}}(k,v)+\widehat{\text{RANK}}(k,v)\right)\right|\\ =&\frac{1}{f_k\hat{f}_k}\cdot\left|\widehat{\text{rank}}(k,v)\cdot\left(\text{RANK}(k,v)-\widehat{\text{RANK}}(k,v)\right) + \widehat{\text{RANK}}(k,v)\cdot\left(\widehat{\text{rank}}(k,v)-\text{rank}(k,v)\right)\right|.\end{aligned}$$

With probability at least $1-2\delta$, $\left|\widehat{\text{rank}}(k,v) - \text{rank}(k,v)\right| \le \varepsilon f_k$ and $\left|\widehat{\text{RANK}}(k,v) - \text{RANK}(k,v)\right| \le \varepsilon f_k$ hold simultaneously. Hence

$$|\hat{q}(k,v) - q(k,v)| \le \frac{1}{f_k \hat{f}_k} \cdot \left(\varepsilon f_k \cdot \widehat{\text{rank}}(k,v) + \varepsilon f_k \cdot \widehat{\text{RANK}}(k,v)\right) = \varepsilon$$

holds with probability at least $1-2\delta$. □

Remark. *For a non-heavy-hitter key k, we can also bound the probability in Equation 4 by*

$$\mathbb{P}\left(\left|\sum_{h=1}^{H-1}\sum_{i=1}^{m_h} X_{i,h}\right| > \theta\varepsilon N\right) \le 2\exp\left(-\frac{\theta^2\varepsilon^2 N^2}{2\sum_{h=1}^{H-1}\sum_{i=1}^{m_h} w_h^2}\right)$$
$$\le 2\exp\left(-\frac{\theta^2\varepsilon^2 N^2}{8N^2/(2c-1)l^2}\right) = 2\exp\left(-\frac{(2c-1)\theta^2\varepsilon^2 l^2}{8}\right).$$

Similarly, by setting $l = \frac{2\sqrt{2}}{\theta\varepsilon\sqrt{2c-1}}\sqrt{\log\frac{2}{\delta}}$, KLL-Polymer ensures that $\left|\widehat{\text{rank}}(k,v) - \text{rank}(k,v)\right| \le \theta\varepsilon N$ with probability at least $1-\delta$, and $\left|\hat{f}_k - f_k\right| \le 2\theta\varepsilon N$ with probability at least $1-2\delta$. However, because this absolute bound can easily exceed the actual frequencies of these infrequent keys, these estimates may become practically trivial. In addition, non-heavy-hitters may not have even a single item retained in the sketch (i.e. $\hat{f}_k = 0$), precluding any estimation entirely. We will further evaluate the actual performance of KLL-Polymer on these infrequent keys in Appendix F in our technical report [39].

It is worth noting that although the basic version of KLL-Polymer can solve the per-key quantile estimation problem, the result of Theorem 2 is far from the desired space complexity of $O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$. We will optimize the space complexity in the next section.

4 Reducing Space Complexity and Keeping Mergeability

Similar to the KLL sketch, several space-reduction techniques can also be applied to KLL-Polymer. We first introduce two such techniques that enable KLL-Polymer to achieve a space complexity of $O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$. We then present the space lower bound for the per-key quantile estimation problem and discuss the gap with our solution. Finally, we show that KLL-Polymer preserves mergeability in the per-key setting and compare its theoretical properties with other per-key algorithms.

4.1 Using a Sampler for Bottom Levels

In KLL-Polymer, level h has a capacity of $\max\left\{\lceil l \cdot c^{H-h}\rceil, 2\right\}$. However, only the top $\left\lfloor\log_c\frac{2}{l}\right\rfloor$ levels have capacity greater than 2; the capacity of the bottom $L = H - \left\lfloor\log_c\frac{2}{l}\right\rfloor$ levels is equal to 2. In this situation, the group-by and sorting operation during compaction is unnecessary: KLL-Polymer just randomly selects one item in this level, doubles its weight, and promotes it to the next level. These levels can be replaced by only one slot using Reservoir Sampling [52].

Optimization 1: Using a Sampler. In the optimized version of KLL-Polymer, the bottom L levels are all removed; instead, we only use one sampler to record one item $e_s = (k_s, v_s)$ and its weight w_s. To insert an item $e = (k,v)$, we first increment w_s by 1. Then, with probability $p = \frac{1}{w_s}$, e successfully replaces e_s and is inserted into the sampler; with probability $p = \frac{w_s-1}{w_s}$, e_s still remains in the sampler. Finally, we check whether the weight w_s has reached 2^L: if so, the item now in the sampler will be inserted into level $L+1$, and we clear the sampler by removing the current item e_s and resetting w_s to 0.

Theorem 5. *After using a sampler for bottom levels, KLL-Polymer achieves per-key quantile estimation with a space complexity of* $O\left(\frac{1}{\theta\varepsilon}\sqrt{\log\frac{1}{\delta}}\right)$.

Proof. Again let $l = \frac{2\sqrt{2}}{\theta\varepsilon\sqrt{2c-1}}\sqrt{\log\frac{2}{\delta}}$. After adding a sampler, the bottom L levels do not store any items now, *i.e.* $c_h = 0$. The top $H - L$ levels still have $c_h = \lceil l \cdot c^{H-h} \rceil$. The overall space complexity now becomes

$$\begin{aligned}\sum_{h=1}^{H} c_h &= 1 + \sum_{l\cdot c^{H-h}\le 2} 0 + \sum_{l\cdot c^{H-h}>2} \lceil l\cdot c^{H-h}\rceil \\ &\le 1 + \sum_{l\cdot c^{H-h}\le 2} \frac{3}{2} l\cdot c^{H-h} + \sum_{l\cdot c^{H-h}>2} \frac{3}{2} l\cdot c^{H-h} \\ &= 1 + \sum_{h=1}^{H} \frac{3}{2} l\cdot c^{H-h} \le 1 + \frac{3l}{2(1-c)} = O\left(\frac{1}{\theta\varepsilon}\sqrt{\log\frac{1}{\delta}}\right).\end{aligned}$$ □

In practice, the bottom-level capacity c_1 easily exceeds 2, making the sampler empirically unnecessary when memory is abundant. Therefore, the fundamental significance of Optimization 1 is strictly theoretical: As the stream size N grows, the sketch height H increases. Without the sampler, the accumulation of these levels would introduce an unavoidable space dependence on N. The sampler mathematically bypasses these layers, eliminating the $\log(\theta\varepsilon N)$ factor from the space complexity. This mechanism is an essential step to ensure that the theoretical space complexity of KLL-Polymer remains independent of the stream length.

4.2 Fixed Capacity for Top Levels

The second observation of KLL-Polymer is also inspired by the KLL sketch. Since items in higher levels carry larger weights, their information should be preserved with higher priority. Therefore, rather than allowing the capacity to diminish exponentially across all levels, we fix the capacity of the top levels to a constant. This design increases the capacity of the high levels compared to a strict geometric decay, ensuring that more high-weight items are kept in KLL-Polymer.

Optimization 2: Fixed Top-Level Capacity. In the optimized version of KLL-Polymer, the capacities of the top s levels are all fixed to $c_h = c_H = l$, *i.e.* they no longer decrease exponentially. The lower $H - s$ levels retain capacities $c_h = \max\left\{\lceil l\cdot c^{H-h}\rceil, 2\right\}$, while the bottom L levels with capacity 2 are replaced by a sampler. The following theorem shows that this modification yields a per-key quantile sketch with reduced space complexity.

Theorem 6. *After setting a fixed capacity for top s levels, KLL-Polymer achieves per-key quantile estimation with a space complexity of* $O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$.

Proof. The key idea is that, when the estimation errors contributed by the bottom $H - s$ levels and the top s levels are both less than $\frac{1}{2}\varepsilon f_k$, their sum is guaranteed to be within εf_k. We now bound the overall estimation error from these two aspects.

For the bottom $H - s$ levels, recalling Equation 3 and setting $H' = H - s - 1$, we have

$$\begin{aligned}&\mathbb{P}\left(\left|\sum_{h=1}^{H-s-1}\sum_{i=1}^{m_h} X_{i,h}\right| > \frac{1}{2}\varepsilon f_k\right) \le 2\exp\left(-\frac{\varepsilon^2 f_k^2}{8\sum_{h=1}^{H-s-1}\sum_{i=1}^{m_h} w_h^2}\right) \\ &\le 2\exp\left\{-\frac{(2c-1)\varepsilon^2 f_k^2 l^2 (2c)^s}{32N^2}\right\} \le 2\exp\left\{-\frac{(2c-1)\theta^2\varepsilon^2 l^2 (2c)^s}{32}\right\}.\end{aligned}$$

To ensure

$$\mathbb{P}\left(\left|\sum_{h=1}^{H-s-1}\sum_{i=1}^{m_h} X_{i,h}\right| > \frac{1}{2}\varepsilon f_k\right) \le \delta,$$

we require $2\exp\left\{-\frac{(2c-1)\theta^2\varepsilon^2 l^2 (2c)^s}{32}\right\} \le \delta$, *i.e.*

$$\theta\varepsilon l \cdot (2c)^{\frac{s}{2}} \ge \sqrt{\frac{32}{2c-1}\log\frac{2}{\delta}}. \tag{5}$$

For the top s levels, their capacity is $c_h = l$ now, and $m_h \le \frac{N}{c_h w_h} \le \frac{N}{l\cdot 2^{h-1}}$,

$$\sum_{h=H-s}^{H-1}\sum_{i=1}^{m_h} w_h = \sum_{h=H-s}^{H-1} m_h w_h \le \frac{sN}{l}.$$

Note that each compaction operation in level h leads to at most w_h error, and the total error of the highest s levels is at most $\frac{sN}{l}$. To ensure this error is smaller than $\frac{1}{2}\varepsilon f_k$, we require

$$2s \le \theta\varepsilon l. \tag{6}$$

Setting $l = \frac{C_1}{\theta\varepsilon}\log\log\frac{1}{\delta}$ and $s = C_2\log\log\frac{1}{\delta}$ for certain constants C_1 and C_2 suffices to satisfy Equations 5 and 6. The overall space complexity is now

$$\sum_{h=1}^{H} c_h \le 1 + \frac{3l}{2(1-c)} + sl = O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right). \qquad \square$$

The proof of Theorem 6 also motivates an additional optimization for KLL-Polymer. Since the error bound of top-level compactions is independent of the randomness assumption, these levels can be compacted *deterministically* rather than *randomly*. Specifically, during each compaction in the top s levels, KLL-Polymer with deterministic compaction (**KLL-Polymer-DC** for short) keeps only $\lfloor\frac{n}{2}\rfloor$ items for each key with n values by always keeping items with even index. In this way, KLL-Polymer-DC still maintains the same worst-case space complexity as the basic version: Hoeffding's Inequality bounds the bottom-level error by $\frac{1}{2}\varepsilon f_k$ with probability at least $1-\delta$, while top-level compactions bound the corresponding error by $\frac{1}{2}\varepsilon f_k$ deterministically. Thus, KLL-Polymer-DC securely retains the $O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$ space bound.

Beyond preserving this strict theoretical baseline, the advantage of KLL-Polymer-DC lies in its practical efficiency. This deterministic policy can directly discard keys that appear only once and reduce the number of items promoted to higher levels, which potentially lowers the space cost for sparse data streams. Particularly in heavy-tailed streams, this filtering shrinks the effective stream size and yields a practical memory footprint smaller than the theoretical bound. However, because quantifying these exact marginal gains depends on the data distribution and the arrival order of items, we do not provide a formal space complexity analysis here; instead, we leave the empirical evaluation of KLL-Polymer-DC to Section 5.3.

In general, the two optimizations of KLL-Polymer can be summarized in Figure 3.

4.3 Space Lower Bound

Here, we present the space lower bound for the per-key quantile estimation problem. We then identify the gap between our solution and this bound, and discuss how it might — or might not — be achieved in practice.

Theorem 7. ***Space Lower Bound.*** *Let $\mathcal{A}_P$ be a randomized comparison-based algorithm that, for an arbitrary data stream S and heavy hitters with threshold θ, solves the ε approximation per-key*

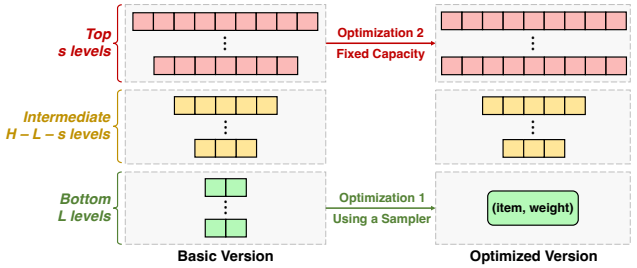


Fig. 3. Two optimizations of KLL-Polymer.

quantile estimation problem with probability at least $1-\delta$. *Then* $\mathcal{A}_P$ *must store at least* $\Omega\left(\frac{1}{\theta\varepsilon}\log\log\frac{1}{\delta}\right)$ *items from the stream.*

Proof. For simplicity let $\theta = \frac{1}{m}$ for some integer m and $m|N$. Consider a data stream consisting of m distinct keys, where k_1 appears $\frac{N}{m}$ times, followed by k_2 appearing $\frac{N}{m}$ times, and so on. In this setting, all m keys are heavy hitters, and $\mathcal{A}_P$ must estimate quantiles for each key. Consequently, the problem reduces to performing single-key quantile estimation independently for each of the m keys. According to [19], single-key quantile estimation requires $\frac{C}{\varepsilon}\log\log\frac{1}{\delta}$ space for some constant C. Thus $\mathcal{A}_P$ requires $m\cdot\frac{C}{\varepsilon}\log\log\frac{1}{\delta} = \frac{C}{\theta\varepsilon}\log\log\frac{1}{\delta}$ space. As a result, the space lower bound for $\mathcal{A}_P$ is $\Omega\left(\frac{1}{\theta\varepsilon}\log\log\frac{1}{\delta}\right)$. □

In fact, if it is allowed to compare two keys in the data stream, then obtaining an algorithm with space complexity $O\left(\frac{1}{\theta\varepsilon}\log\log\frac{1}{\delta}\right)$ becomes straightforward: Define the ordering $(k_1,v_1)\prec(k_2,v_2)$ if $k_1<k_2$, or $k_1=k_2$ and $v_1<v_2$. One can simply construct a KLL sketch with $\varepsilon'=\frac{\theta\varepsilon}{2}$ and $\delta'=\frac{\delta}{2}$, and then compute $\widehat{\mathrm{rank}}'(k,v)-\widehat{\mathrm{rank}}'(k,-\infty)$ from the new KLL sketch as $\widehat{\mathrm{rank}}(k,v)$, where $-\infty$ denotes the minimum value in the data stream. Since

$$\begin{aligned}
&\left|\widehat{\mathrm{rank}}(k,v)-\mathrm{rank}(k,v)\right|\\
=&\left|\left(\widehat{\mathrm{rank}}'(k,v)-\mathrm{rank}'(k,v)\right)-\left(\widehat{\mathrm{rank}}'(k,-\infty)-\mathrm{rank}'(k,-\infty)\right)\right|\\
\le&\left|\widehat{\mathrm{rank}}'(k,v)-\mathrm{rank}'(k,v)\right|+\left|\widehat{\mathrm{rank}}'(k,-\infty)-\mathrm{rank}'(k,-\infty)\right|\\
\le&2\varepsilon' N=\varepsilon\cdot\theta N\le\varepsilon f_k,
\end{aligned}$$

with probability at least $1-2\delta'=1-\delta$, this reduction is valid under the assumption that the keys can be totally ordered, and the space cost is $O\left(\frac{1}{\varepsilon'}\log\log\frac{1}{\delta'}\right)=O\left(\frac{1}{\theta\varepsilon}\log\log\frac{1}{\delta}\right)$.

However, in our problem setting, keys are treated as categorical identifiers rather than numeric values. Without a natural total order among keys, algorithms cannot rely on sorting keys to gain additional information.[4] In this context, Theorem 7 leaves a potential gap of $\log\log\frac{1}{\delta}$ between KLL-Polymer and the space lower bound. In the single-key situation, the KLL sketch eliminates this $\log\log\frac{1}{\delta}$ factor by inserting items in the top level into a GK sketch. In contrast, in the per-key setting, the number of heavy hitters is not known in advance, so it is not possible to pre-allocate GK sketches to save space. Furthermore, values corresponding to different keys cannot be mixed, and thus cannot be inserted into the same GK sketch. Consequently, designing a per-key quantile estimation algorithm with $O\left(\frac{1}{\theta\varepsilon}\log\log\frac{1}{\delta}\right)$ space complexity is highly challenging.

[4]Note that although KLL-Polymer applies a group-by operation during compaction, this does not rely on comparing keys. It only requires that items with the same key be contiguous; no total order among keys is assumed or required.

4.4 Mergeability

Mergeability is an important property of the KLL sketch. Its official definition is stated as follows [19]:

Definition 4. *Let $\mathcal{A}$ be a sketching algorithm. For two data streams S_1 and S_2 and two sketches $\mathcal{A}_{S_1}$ and $\mathcal{A}_{S_2}$ summarizing S_1 and S_2 with the algorithm respectively, if there exists a merge operation to construct a new sketch $\mathcal{A}_m$, s.t. $\mathcal{A}_m$ achieves the same theoretical error guarantees as the sketch $\mathcal{A}_{S_1 \cup S_2}$ directly built on $S_1 \cup S_2$, then the sketching algorithm $\mathcal{A}$ is said to be* ***mergeable****.*

The basic version of KLL-Polymer is mergeable: To merge two sketches, KLL-Polymer simply inserts all items from the smaller sketch (*i.e.* the one summarizing fewer items) into the corresponding levels of the larger one. Whenever the number of items in level h exceeds its capacity c_h, KLL-Polymer applies the usual group-by and compaction operation to the first c_h items, promoting half of them to the next level. This compaction process continues until all levels satisfy their capacity constraints, yielding a valid merged sketch.

KLL-Polymer with two optimizations is also mergeable: During the merging operation, KLL-Polymer just checks the items and their weights maintained in each sampler. Suppose the items in the larger and the smaller sketch are $e_s = (k_s, v_s)$ and $e' = (k'_s, v'_s)$, and their associated weights are w_s and w'_s respectively. Following the strategy of the KLL sketch, if $w_s + w'_s \leq 2^L$ (recall that L denotes the number of levels with capacity 2), then e'_s will evict e_s with probability $\frac{w'_s}{w_s + w'_s}$. Regardless of whether the eviction occurs, KLL-Polymer sets the weight of the item that remains in the sampler to $w_s + w'_s$, and if the weight $w_s + w'_s$ is now 2^L, KLL-Polymer inserts the item in the sampler into level $L + 1$ and clears the sampler; If $w_s + w'_s > 2^L$, KLL-Polymer just keeps the item with smaller weight in the sampler, and modifies its weight to $\min\{w_s, w'_s\}$. With probability $\frac{\max\{w_s, w'_s\}}{2^L}$, KLL-Polymer promotes the item with larger weight into level $L + 1$. In addition, since the optimization of setting fixed capacity for the top s levels does not require any modification to the insertion procedure, this version of KLL-Polymer is also mergeable.

However, KLL-Polymer with deterministic compaction is not mergeable: The crucial issue is that lower levels must preserve the i.i.d. structure to apply Hoeffding's Inequality. However, while the deterministic compaction is applied only to the *top* levels, two sketches of different sizes may disagree on which levels count as *top*. As a result, items that were deterministically compacted in the smaller sketch's top levels may be inserted into the larger sketch's bottom levels during merging, thereby violating the independence assumptions needed for the error analysis. Hence, KLL-Polymer-DC is not mergeable. Thus, we position it as an optional practical variant: the standard KLL-Polymer is recommended for distributed, merge-heavy pipelines, whereas KLL-Polymer-DC is preferred for single-node monitoring where its superior empirical accuracy outweighs the need for mergeability.

4.5 Analysis and Discussion

Finally, we compare the properties of KLL-Polymer with other per-key quantile estimation algorithms, *e.g.* HistSketch [33], SQUAD [34, 35], SketchPolymer [36] and M4 [37, 38].

Regarding space cost, SQUAD proposes a sampling method with a space complexity of $O\left(\frac{1}{\theta\varepsilon^2}\log\frac{1}{\delta}\right)$, and a sketching method with a space complexity of $O\left(\frac{1}{\theta\varepsilon^2}\log(\theta\varepsilon^2 N)\right)$. Finally, SQUAD combines both methods to achieve an overall complexity of $O\left(\frac{1}{\theta\varepsilon^{1.5}}\log\frac{1}{\delta}\right)$. Ignoring the logarithm factors, the space complexity of SQUAD is $\widetilde{O}\left(\theta^{-1}\varepsilon^{-1.5}\right)$, which is significantly worse than KLL-Polymer of $\widetilde{O}\left(\theta^{-1}\varepsilon^{-1}\right)$.

Other algorithms (*i.e.* HistSketch, SketchPolymer and M4) are not necessarily comparison-based, making their space complexity difficult to analyze under this setting. As to HistSketch, let $f_{k,I}$ denote the frequency for a key k with values in interval I, and $\widehat{f_{k,I}}$ denote its estimated frequency. HistSketch guarantees that $\left|\widehat{f_{k,I}} - f_{k,I}\right| \leq \varepsilon N$ with probability at least $1-\delta$ using $O\left(\frac{1}{\varepsilon}\log\frac{1}{\delta}\right)$ space. However, HistSketch does not give an explicit error bound for per-key quantile estimation. In addition, HistSketch divides all values into a fixed number of intervals in advance, so its error bound also relies on value distribution. SketchPolymer states that, under certain assumptions, it achieves

$$\mathbb{P}\left(\left|\widehat{\text{rank}}(k,v) - \text{rank}(k,v)\right| \geq \varepsilon N\right) \leq C\left(\frac{N}{\varepsilon f_k w}\right)^d,$$

where d and w denote the depth and the width of the Count-Min Sketch used in Stage 2 and Stage 3, and C is a constant. By setting $w = \frac{C_1}{\theta\varepsilon}$ and $d = C_2 \log\frac{1}{\delta}$ for sufficiently large constants C_1 and C_2, the failure probability becomes at most δ. This parameter choice implies a space cost of $2dw = O\left(\frac{1}{\theta\varepsilon}\log\frac{1}{\delta}\right)$, which is less efficient than KLL-Polymer even in non–comparison-based settings. M4 states that, if m distinct keys are hashed into n buckets in its highest level using w hash functions, then the collision probability for key k is at most $e^{-\frac{mw}{n}}$. Combining this with a single-key algorithm META with failure probability $\frac{\delta}{2}$ in each bucket and requiring $e^{-\frac{mw}{n}} \leq \frac{\delta}{2}$ yields a per-key solution with failure probability δ. This implies a top-level space cost of $mw\log\frac{2}{\delta} \times \text{Space(META)}$, where Space(META) denotes the space complexity of META. However, M4 also maintains the information of infrequent keys in lower levels, so its real space cost is considerably higher; moreover, m depends heavily on data distribution, making the exact complexity hard to compute.

In terms of mergeability, the data structure of HistSketch is based on mergeable Elastic Sketch [56], so HistSketch is mergeable. The sampling technique (Reservoir Sampling) and sketching technique (Space Saving) used by SQUAD are all mergeable [57]. Therefore, SQUAD is mergeable. Since two key components of SketchPolymer, *i.e.* Count-Min Sketch and Bloom filter, are both mergeable, SketchPolymer is mergeable as well. Finally, as M4 usually applies mergeable algorithms as META to summarize buckets from the same level, M4 is also mergeable.

In general, we show the properties of different per-key algorithms in Appendix A in our technical report [39].

5 Experimental Results

In this section, we present a comprehensive experimental evaluation of KLL-Polymer and its variants. We introduce our experimental setup in Section 5.1. Then we compare the performance of KLL-Polymer with prior work in Section 5.2. We further demonstrate the effect of deterministic compaction in Section 5.3. We analyze the performance of KLL-Polymer from several aspects in Section 5.4. Finally, we implement KLL-Polymer in the RocksDB database to show its practical deployment in Section 5.5. The source code is available on GitHub [39].

5.1 Experimental Setup

Implementation: We implement KLL-Polymer and all other algorithms in C++, and use Bob Hash [58] with different hash seeds to implement the hash functions. All experiments are conducted on a server with one 18-core processor (36 threads, Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz) and 128 GB DRAM memory. The processor has 64KB L1 cache, 1MB L2 cache for each core, and 24.75MB L3 cache shared by all cores.

Datasets: We use six datasets in our experiments, including two real datasets and four synthetic datasets. A table for detailed information of each dataset is shown in Appendix B in our technical report [39].

- **CAIDA Dataset:** This dataset consists of streams of anonymized IP traces collected by CAIDA [59]. Each packet (item) is identified by a five-tuple: source and destination IP addresses, source and destination ports, and protocol, together with a timestamp. We regard the source and destination IP addresses of a packet as its key, and the timestamp as its value. We use 20M items, with 9,417,590 distinct keys. We set $\theta = 10^{-4}$, and 298 keys are classified as heavy hitters.
- **MAWI Dataset:** This dataset consists of real traffic traces provided by the MAWI Working Group [60]. Each flow (item) is represented by a flow ID and a timestamp. We regard the flow ID as its key, and the timestamp as its value. We use 10M items, with 2,173,066 distinct keys. We set $\theta = 10^{-3}$, and 45 keys are classified as heavy hitters.
- **Synthetic Dataset:** We generate four synthetic datasets with keys following the Uniform distribution (1K keys) and the Zipfian distribution with parameter $\alpha = 1.5$ (100K keys), and values following the Gamma distribution and the Lognormal distribution.[5] We use 10M items, and set $\theta = 10^{-3}$.

Comparison Algorithms: We conduct experiments to compare the performance of KLL-Polymer with four baselines, *i.e.* HistSketch [33], SQUAD [34, 35], SketchPolymer [36] and M4 [37, 38]. Regarding the parameter setting of KLL-Polymer, its memory cost can be approximated as follows:

$$\text{Space} \approx \sum_{h=1}^{H-s-1} l \cdot c^{H-s-h} + s \cdot l \approx l\left(s + \frac{1}{1-c}\right). \tag{7}$$

We fix $c = \frac{2}{3}$ and $s = 3$, and compute the value of l according to Equation 7. In addition, the height of KLL-Polymer satisfies Equation 2, so

$$H \geq \log \frac{N}{l} + 1 \Rightarrow H_{\min} = \left\lceil \log \frac{N}{l} \right\rceil + 1.$$

Due to space limits, the parameter settings of other algorithms are shown in Appendix C in our technical report [39].

Metrics: Per-key algorithms usually support rank queries (given a key k and a value v, return $\widehat{\text{rank}}(k, v)$) and quantile queries (given a key k and a quantile $w \in (0, 1)$, return a value v with $\hat{q}(k, v) = w$). For rank queries, we find all values with quantiles $\{1\%, 2\%, \cdots, 99\%\}$ for heavy hitters, and query their rank. For quantile queries, we query quantiles $\{1\%, 2\%, \cdots, 99\%\}$ for heavy hitters. We use the following metrics to measure the performance of these algorithms:

- **Average Rank Error (ARE):** This metric is used for rank queries. Suppose we perform rank queries for heavy hitters $k_1, \cdots, k_n$ and values $v_{i,1}, \cdots, v_{i,99}$, $i = 1, \cdots, n$, Average Rank Error (ARE) is defined as

$$\text{ARE} = \frac{1}{99n} \sum_{i=1}^{n} \sum_{j=1}^{99} \left| \widehat{\text{rank}}(k_i, v_{i,j}) - \text{rank}(k_i, v_{i,j}) \right|.$$

- **Average Quantile Error (AQE):** This metric is used for quantile queries. Suppose we perform quantile queries for heavy hitters $k_1, \cdots, k_n$, and the algorithm returns $\tilde{v}_{i,j}$ as the $j\%$ quantile of key k_i. Average Quantile Error (AQE) is defined as

$$\text{AQE} = \frac{1}{99n} \sum_{i=1}^{n} \sum_{j=1}^{99} \left| q(k_i, \tilde{v}_{i,j}) - j\% \right|.$$

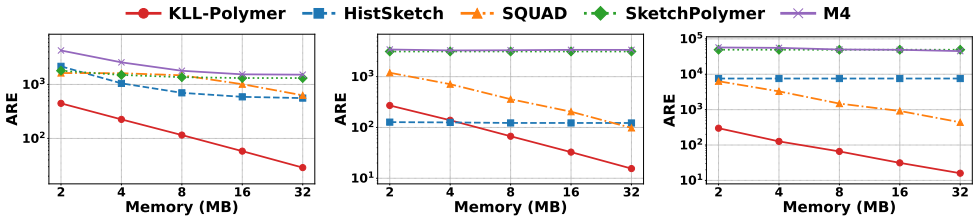
[5] Throughout the paper, we use the notation *A-B* to denote a synthetic dataset with keys following distribution *A*, and values following distribution *B*, *e.g.* Uniform-Gamma refers to the dataset with uniformly distributed keys and Gamma-distributed values.

- **Throughput:** We use millions of operations (insertions) per second (Mops) to measure the throughput. We repeat the experiment 10 times and calculate the average results as the throughput. To ensure we capture steady-state performance, these measurements are recorded after an initial warm-up phase.

5.2 Comparison with Prior Work

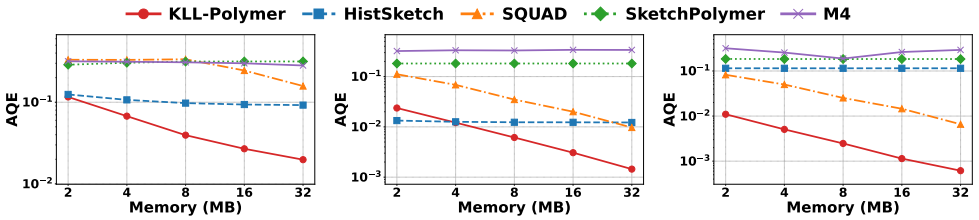
In this section, we compare the performance of KLL-Polymer with prior work across six datasets. Due to space constraints, we report detailed results for the CAIDA, Uniform-Gamma and Zipfian-Lognormal datasets in the main paper. The remaining results on the MAWI, Uniform-Lognormal and Zipfian-Gamma datasets are provided in Appendix D in our technical report [39].

5.2.1 Accuracy: KLL-Polymer achieves significantly lower error than state-of-the-art algorithms. As shown in Figure 4, KLL-Polymer reduces ARE by 10.15×, 3.81× and 244.51× compared to HistSketch, and 17.39×, 6.38× and 29.58× compared to SQUAD on the CAIDA, Uniform-Gamma and Zipfian-Lognormal datasets using 16MB memory. Similar results are observed in Figure 5, where KLL-Polymer outperforms HistSketch by 3.48×, 4.01× and 82.07×, and SQUAD by 9.10×, 6.53× and 12.76× in terms of AQE.



(a) CAIDA (b) Uniform-Gamma (c) Zipfian-Lognormal

Fig. 4. Average Rank Error (ARE) on different datasets.



(a) CAIDA (b) Uniform-Gamma (c) Zipfian-Lognormal

Fig. 5. Average Quantile Error (AQE) on different datasets.

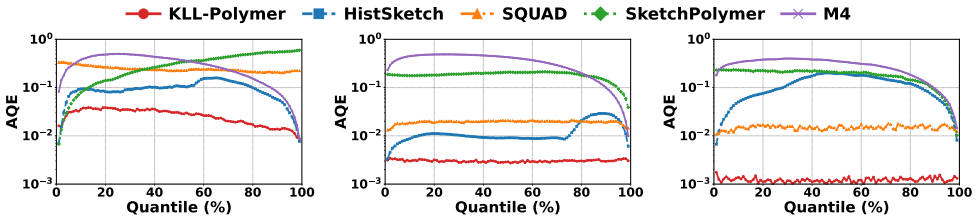
To better demonstrate the advantage of KLL-Polymer, we measure the AQE across different quantiles within 16MB memory, *i.e.* for $j \in \{1, 2, \cdots, 99\}$, we calculate

$$\mathrm{AQE}(j\%) = \frac{1}{n}\sum_{i=1}^{n}\left|q(k_i, \tilde{v}_{i,j}) - j\%\right|$$

and show this result in Figure 6. The experimental results demonstrate that KLL-Polymer achieves consistently low and stable error. The AQE across all query quantiles is less than 0.04 on the CAIDA dataset, and less than 0.01 on the Uniform-Gamma and Zipfian-Lognormal datasets. While SQUAD also witnesses stable AQE, its error is much higher than that of KLL-Polymer.

We also observe that non-comparison-based algorithms typically suffer from a bottleneck caused by fixed interval partitioning. HistSketch, SketchPolymer and M4 perform best only when values

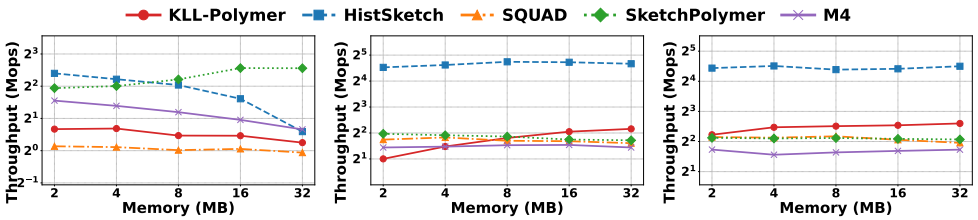
are evenly distributed across many intervals, and they incur significant error when diverse values cluster within a single interval. Consequently, their accuracy is fundamentally capped by the chosen interval granularity, sometimes leading to stagnant performance even as memory increases. However, comparison-based algorithms like KLL-Polymer capture precise distributions without any prior knowledge of the value, making them highly robust to diverse and unknown data distributions.



(a) CAIDA (b) Uniform-Gamma (c) Zipfian-Lognormal

Fig. 6. Average Quantile Error (AQE) on different quantiles.

5.2.2 Speed: We list the throughput of different algorithms in Figure 7, and find that comparison-based algorithms, including KLL-Polymer, generally run slower than non-comparison-based algorithms. Since the quantile estimation problem reduces to the frequency estimation problem over different intervals, HistSketch, SketchPolymer and M4 utilize existing frequency estimation sketches to achieve high insertion throughput. In contrast, comparison-based algorithms must preserve item ordering to provide rigorous guarantees, which leads to additional computational overhead. Nonetheless, KLL-Polymer also demonstrates superior throughput within the comparison-based category: the throughput of KLL-Polymer is 1.33×, 1.29× and 1.39× higher than that of SQUAD on the CAIDA, Uniform-Gamma and Zipf-Lognormal datasets respectively. Furthermore, the throughput of KLL-Polymer exhibits complex fluctuations across varying memory budgets, as the compaction overhead is highly sensitive to the key distribution within each specific level.



(a) CAIDA (b) Uniform-Gamma (c) Zipfian-Lognormal

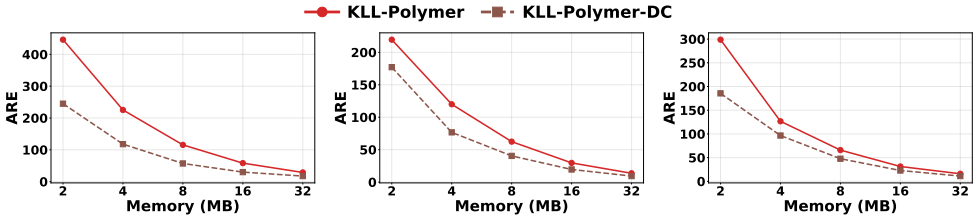
Fig. 7. Throughput on different datasets.

5.3 Impact of Deterministic Compaction

Here, we evaluate the performance of KLL-Polymer-DC in comparison to the basic KLL-Polymer. In the standard KLL-Polymer, its height H can be analytically determined by $H = \lceil \log \frac{N}{I} \rceil + 1$. However, for KLL-Polymer-DC, the total weight of items (*i.e.* the effective N_{DC}) is difficult to obtain in advance due to the deterministic compaction strategy. In the absence of an explicit expression for N_{DC}, we set $H_{DC} = \lceil \log \frac{N}{I} \rceil$ based on empirical observations.

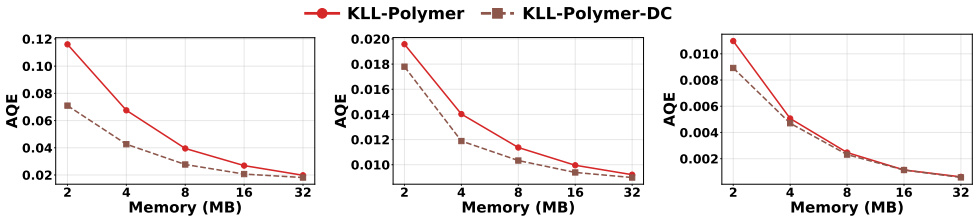
Figures 8 and 9 present the ARE and AQE performance of KLL-Polymer-DC across three datasets. As to the CAIDA dataset, KLL-Polymer-DC achieves 1.82× lower ARE and 1.63× lower AQE compared to KLL-Polymer under a 2MB memory budget, which highlights the effects of deterministic

compaction especially in memory-constrained environments. Similar trends are observed on the MAWI and Zipfian-Lognormal datasets, where KLL-Polymer-DC reduces error by 1.23×/1.10× and 1.61×/1.23× respectively. Furthermore, the accuracy advantage of deterministic compaction persists as the memory allocation scales, consistently yielding smaller error than the basic version even under larger memory budgets. We also plot the throughput in Figure 10. As shown, both KLL-Polymer and KLL-Polymer-DC exhibit comparable throughput performance. However, due to a lower sketch height and larger per-level capacities in KLL-Polymer-DC, their dynamic key distributions and compaction triggers differ fundamentally, which makes a straightforward, deterministic analysis of their throughput highly complex.



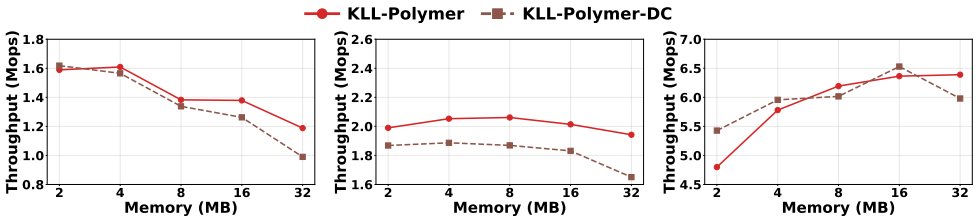
(a) CAIDA (b) MAWI (c) Zipfian-Lognormal

Fig. 8. Average Rank Error (ARE) of deterministic compaction on different datasets.



(a) CAIDA (b) MAWI (c) Zipfian-Lognormal

Fig. 9. Average Quantile Error (AQE) of deterministic compaction on different datasets.



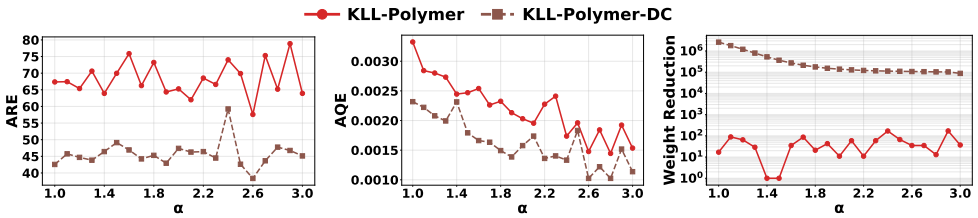
(a) CAIDA (b) MAWI (c) Zipfian-Lognormal

Fig. 10. Throughput of deterministic compaction on different datasets.

To better demonstrate the impact of deterministic compaction, we generate different Zipfian-Lognormal datasets with Zipfian parameter $\alpha \in [1.0, 3.0]$. We run KLL-Polymer and KLL-Polymer-DC on the generated datasets, and the results of ARE, AQE and weight reduction[6] within 8MB memory are illustrated in Figure 11. The experimental results demonstrate that both KLL-Polymer and KLL-Polymer-DC deliver consistently high accuracy across varying degrees of skewness, with

[6]Here, weight reduction refers to the difference between the stream size N and the total weight of items retained in the sketch (*i.e.* N_{DC}).

ARE smaller than 80 and AQE smaller than 0.004. Furthermore, KLL-Polymer-DC effectively filters infrequent keys and achieves smaller error than KLL-Polymer. In heavy-tailed scenarios when α is small, KLL-Polymer-DC filters up to 30% items in the data stream. Such substantial reduction of N significantly decreases the frequency of compaction operation at top levels, and even potentially leads to a smaller sketch height H in boundary cases. Therefore, this minimizes the cumulative compaction error, as fewer compactions introduce less variance. In contrast, the weight reduction in KLL-Polymer is merely incidental rather than strategic. Such fluctuations only occur when c_h of level h is odd, and even then, the resulting weight loss is bounded by at most 2^{h-1}. Consequently, KLL-Polymer fails to achieve significant weight reduction, resulting in ineffectiveness *w.r.t.* filtering infrequent keys and larger estimation error compared to KLL-Polymer-DC.



(a) Average Rank Error (ARE) (b) Average Quantile Error (AQE) (c) Weight Reduction

Fig. 11. Effects of deterministic compaction on Zipfian-Lognormal dataset.

5.4 Analysis of KLL-Polymer

In this section, we conduct a comprehensive performance analysis of KLL-Polymer. We first investigate the robustness of KLL-Polymer by analyzing the impact of different input arrival orders. Next, we evaluate how the skewness of the underlying data distribution affects the overall throughput. Finally, we provide a fine-grained micro-benchmark profiling to break down the total insertion latency among the core operations.

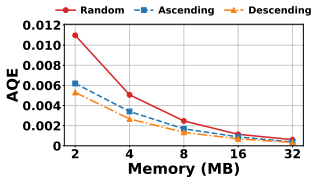


Fig. 12. Effects of input orderings.

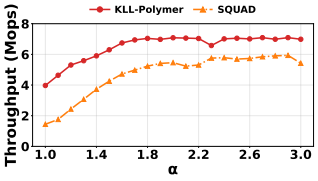


Fig. 13. Throughput vs. skewness.

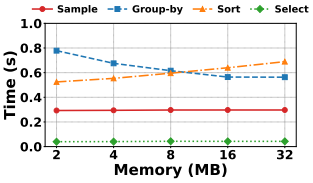


Fig. 14. Insertion time breakdown.

Effects of input orders: To evaluate the sensitivity of KLL-Polymer to data arrival patterns, we employ the Zipfian-Lognormal dataset and construct three extreme input orderings: purely ascending by value, purely descending by value, and completely random. As depicted in Figure 12, KLL-Polymer achieves consistently high estimation accuracy regardless of the input sequence. Even under a tightly constrained memory budget of 2MB, the AQE across all three orderings remains below 0.012. As the memory budget scales up to 32MB, the estimation errors converge to nearly identical minimal values, which demonstrates the algorithmic robustness of KLL-Polymer and the resilience against worst-case inputs in the data stream.

Effects of data skewness: We measure the insertion throughput of KLL-Polymer and SQUAD using different Zipfian-Lognormal datasets with Zipfian parameter $\alpha \in [1.0, 3.0]$. As illustrated in Figure 13, the throughput of both algorithms increases as α grows. This upward trend is highly

intuitive: a larger α indicates a more highly skewed stream, which diminishes the proportion of items with infrequent keys. A reduction in such items lowers the group-by operation overhead during compaction, thereby yielding a higher overall throughput. Furthermore, the experimental results demonstrate that KLL-Polymer consistently outperforms SQUAD, validating its performance advantage among comparison-based sketch algorithms.

Breakdown of insertion time: To gain a deeper micro-architectural insight, we explicitly decompose the total insertion latency into four constituent operations: Reservoir Sampling, group-by, sorting, and odd/even selection. The fine-grained insertion time breakdown is presented in Figure 14. As the results illustrate, the execution times for Reservoir Sampling and odd-even selection remain remarkably stable across varying memory budgets. In contrast, the latencies for group-by and sorting exhibit a clear trade-off driven by the memory configuration. As the memory budget increases, the capacity of each level expands. Since sorting n items incurs an $O(n \log n)$ complexity, the sorting latency steadily increases. Conversely, the expanded capacity reduces the total number of compactions, thereby effectively bringing down the time cost of the group-by operations.

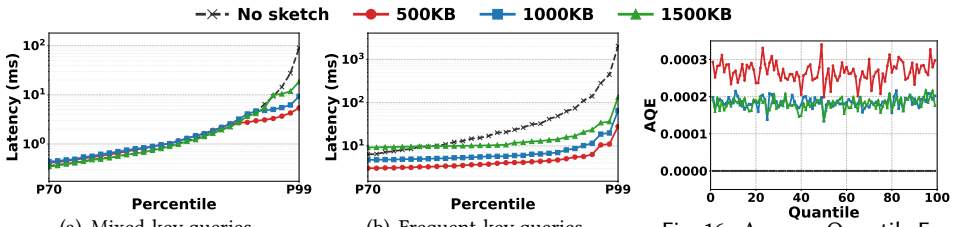
5.5 Application: RocksDB Integration

Lastly, we integrate KLL-Polymer into the RocksDB database to demonstrate its practical utility in accelerating per-key quantile estimation within production-scale storage engines.

RocksDB [61] is a high-performance persistent key-value store built on a Log-Structured Merge-tree (LSM-tree) architecture. It optimizes write performance by buffering incoming records in memory-resident MemTables. Once these buffers reach a predefined size threshold, RocksDB flushes the data into immutable Sorted String Tables (SSTables) on disk. To minimize the cost of accessing multiple SSTables during point queries (*i.e.*, querying a specific key), RocksDB implements a Bloom Filter [48] for each SSTable. RocksDB only initiates a disk-read for a key lookup if the filter indicates that the target key potentially exists within that specific file. However, RocksDB lacks native support for per-key quantile estimation. Calculating quantiles for a key k necessitates finding all associated records, which incurs significant read amplification and latency when its frequency f_k is high.

Implementation: Since RocksDB stores unique keys, we append random suffixes to items to allow multiple occurrences. By configuring `prefix_extractor`, RocksDB identifies items with the same key prefix, while `prefix_same_as_start` ensures retrieval of all records for a specific key during quantile queries. To accelerate per-key quantile estimation, we integrate KLL-Polymer within each SSTable. We also maintain a global Count-Min Sketch [47] to support fast frequency estimation. To answer a quantile query for a key k, we first query the Count-Min Sketch to check whether k is a heavy hitter: if so, we query the Bloom Filter of each SSTable, retrieve a lightweight vector containing all values and their corresponding weights associated with key k from each relevant sketch, and merge these vectors to obtain the final quantile; otherwise, we directly read all records associated with k from the database and compute the exact quantile from these values.

Experiments: We generate the Zipfian-Lognormal dataset with 64-bit keys and 64-bit values containing 10M items, and append a 64-bit suffix to each key to differentiate items with the same key in RocksDB. We create two quantile query workloads, one for mixed keys (keys with $f_k \geq 150$) and one for frequent keys (keys with $f_k \geq 2000$), and measure the tail latency for these workloads to evaluate the performance gains of KLL-Polymer with varying sketch sizes over the baseline (*i.e.* without quantile sketch). The experimental results (Figure 15) demonstrate that KLL-Polymer is effective even for mixed-key queries. While the P70-P90 latency increases by less than 0.2ms, the P99 latency is reduced by 16.77×/9.73×/4.89× when using 500/1000/1500KB per sketch respectively. For frequent-key queries, the P99 latency is reduced by 71.88×/31.28×/15.14× when



(a) Mixed-key queries (b) Frequent-key queries

Fig. 15. Tail latency on quantile queries.

Fig. 16. Average Quantile Error (AQE) on frequent-key queries.

using 500/1000/1500KB per sketch respectively. Figure 16 also proves that KLL-Polymer introduces minimal error overhead: the AQE remains within 0.0005 even using 500KB per sketch, providing high-precision distribution summaries that are robust across all query quantiles.

6 Conclusion

In this paper, we propose KLL-Polymer, a near-optimal quantile sketch for per-key estimation in streaming models. KLL-Polymer preserves the accuracy of individual keys by incorporating a group-by operation into the hierarchical compaction process, and inherits two optimizations from the KLL sketch to achieve a space complexity of $O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$, which closely matches the space lower bound of $O\left(\frac{1}{\theta\varepsilon}\log\log\frac{1}{\delta}\right)$ derived in our analysis. To better handle sparse data streams in real-world scenarios, we further develop a deterministic-compaction variant, KLL-Polymer-DC. The experimental results highlight that KLL-Polymer reduces quantile estimation error by up to 29.58× compared to existing state-of-the-art solutions. We also show its versatility by integrating KLL-Polymer into RocksDB, where it significantly reduces tail latency for per-key quantile queries in real databases.

Acknowledgments

We would like to thank Qilong Shi and Yuanpeng Li for their helpful discussions. We sincerely thank the anonymous reviewers for their constructive comments. This work was supported in part by the National Key Research and Development Program of China under Grant No. 2024YFB2906602, in part by the National Natural Science Foundation of China (NSFC) under Grant No. 62372009 and Grant No. 62402012, and in part by the Basic and Frontier Research Project of Pengcheng Laboratory under Grant No. 2025QYB004.

References

[1] Miklós Csörgő. *Quantile processes with statistical applications*. SIAM, 1983.

[2] Emanuel Parzen. Quantile probability and statistical data modeling. *Statistical Science*, pages 652–662, 2004.

[3] Gilbert W Bassett Jr, Mo-Yin S Tam, and Keith Knight. Quantile models and estimators for data analysis. *Metrika*, 55(1):17–26, 2002.

[4] Robert A Lodder and Gary M Hieftje. Quantile analysis: a method for characterizing data distributions. *Applied spectroscopy*, 42(8):1512–1520, 1988.

[5] Zhiwei Chen and Aoqian Zhang. A survey of approximate quantile computation on large-scale data. *IEEE Access*, 8:34585–34597, 2020.

[6] Yuheng Zhou, Guoju Gao, Yu-E Sun, He Huang, Yang Du, and Yihuai Wang. Piqsketch: An efficient sketching algorithm for per-key tail quantile estimation in large-scale data streams. In *International Conference on Advanced Data Mining and Applications*, pages 311–326. Springer, 2024.

[7] John M Chambers, David A James, Diane Lambert, and Scott Vander Wiel. Monitoring networked applications with incremental quantile estimation. *Statistical Science*, 21(4):463–475, 2006.

[8] Ebenezer RHP Isaac and Bulbul Singh. Qbsd: Quartile-based seasonality decomposition for cost-effective ran kpi forecasting. In *2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS)*, pages 847–851. IEEE, 2025.

[9] Petra Ristau and Lukas Krain. Real time financial risk monitoring as a data-intensive application. In *CLOSER*, pages 660–665, 2019.

[10] Paris Carbone, Gábor E Gévay, Gábor Hermann, Asterios Katsifodimos, Juan Soto, Volker Markl, and Seif Haridi. Large-scale data stream processing systems. In *Handbook of big data technologies*, pages 219–260. Springer, 2017.

[11] Fuheng Zhao, Punnal Ismail Khan, Divyakant Agrawal, Amr El Abbadi, Arpit Gupta, and Zaoxing Liu. Panakos: Chasing the tails for multidimensional data streams. *Proceedings of the VLDB Endowment*, 16(6):1291–1304, 2023.

[12] Jiarui Guo, Boxuan Chen, Kaicheng Yang, Tong Yang, Zirui Liu, Qiuheng Yin, Sha Wang, Yuhan Wu, Xiaolin Wang, Bin Cui, Tao Li, Xi Peng, Renhai Chen, and Gong Zhang. Hourglasssketch: An efficient and scalable framework for graph stream summarization. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 114–127, 2025.

[13] Aleksander Łukasiewicz, Jakub Tětek, and Pavel Veselỳ. Splinesketch: Even more accurate quantiles with error guarantees. *arXiv preprint arXiv:2504.01206*, 2025.

[14] Elena Gribelyuk, Pachara Sawettamalya, Hongxun Wu, and Huacheng Yu. Simple & optimal quantile sketch: Combining greenwald-khanna with khanna-greenwald. *Proceedings of the ACM on Management of Data*, 2(2):1–25, 2024.

[15] Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. Moment-based quantile sketches for efficient high cardinality aggregation queries. *arXiv preprint arXiv:1803.01969*, 2018.

[16] Monica Chiosa, Thomas B Preußer, and Gustavo Alonso. Skt: A one-pass multi-sketch data analytics accelerator. *Proceedings of the VLDB Endowment*, 14(11):2369–2382, 2021.

[17] Fuheng Zhao, Divyakant Agrawal, Amr El Abbadi, and Ahmed Metwally. Spacesaving±: An optimal algorithm for frequency estimation and frequent items in the bounded deletion model. *arXiv preprint arXiv:2112.03462*, 2021.

[18] Zirui Liu, Chaozhe Kong, Kaicheng Yang, Tong Yang, Ruijie Miao, Qizhi Chen, Yikai Zhao, Yaofeng Tu, and Bin Cui. Hypercalm sketch: One-pass mining periodic batches in data streams. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 14–26. IEEE, 2023.

[19] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 71–78. IEEE Computer Society, 2016.

[20] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.

[21] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 207–222, 2021.

[22] Weihe Li and Paul Patras. Stable-sketch: A versatile sketch for accurate, fast, web-scale data stream processing. In *Proceedings of the ACM Web Conference 2024*, pages 4227–4238, 2024.

[23] Yikai Zhao, Wenchen Han, Zheng Zhong, Yinda Zhang, Tong Yang, and Bin Cui. Double-anonymous sketch: Achieving top-k-fairness for finding global top-k frequent items. *Proceedings of the ACM on Management of Data*, 1(1):1–26, 2023.

[24] Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2552–2566, 2017.

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

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

[27] Bo Ji, Changhee Joo, and Ness Shroff. Throughput-optimal scheduling in multihop wireless networks without per-flow information. *IEEE/ACM Transactions On Networking*, 21(2):634–647, 2012.

[28] Kevin Zhao, Prateesh Goyal, Mohammad Alizadeh, and Thomas E Anderson. Scalable tail latency estimation for data center networks. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 685–702, 2023.

[29] Chunquan Liang, Yang Zhang, Yanming Nie, and Shaojun Hu. Continuously maintaining approximate quantile summaries over large uncertain datasets. *Information Sciences*, 456:174–190, 2018.

[30] Michael B Greenwald and Sanjeev Khanna. Quantiles and equi-depth histograms over streams. In *Data Stream Management: Processing High-Speed Data Streams*, pages 45–86. Springer, 2016.

[31] Yuhan Wu, Aomufei Yuan, Zhouran Shi, Yuanpeng Li, Yikai Zhao, Peiqing Chen, Tong Yang, and Bin Cui. Online detection of outstanding quantiles with quantilefilter. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 831–844. IEEE, 2024.

[32] Alberto Miguel-Diez, Adrián Campazas-Vega, Ángel Manuel Guerrero-Higueras, Claudia Álvarez-Aparicio, and Vicente Matellán-Olivera. Anomaly detection in network flows using unsupervised online machine learning. *arXiv preprint arXiv:2509.01375*, 2025.

[33] Jintao He, Jiaqi Zhu, and Qun Huang. Histsketch: A compact data structure for accurate per-key distribution monitoring. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 2008–2021. IEEE, 2023.

[34] Rana Shahout, Roy Friedman, and Ran Ben Basat. Squad: combining sketching and sampling is better than either for per-item quantile estimation. In *Proceedings of the 15th ACM International Conference on Systems and Storage*, pages 152–152, 2022.

[35] Rana Shahout, Roy Friedman, and Ran Ben Basat. Together is better: Heavy hitters quantile estimation. *Proceedings of the ACM on Management of Data*, 1(1):1–25, 2023.

[36] Jiarui Guo, Yisen Hong, Yuhan Wu, Yunfei Liu, Tong Yang, and Bin Cui. Sketchpolymer: Estimate per-item tail quantile using one sketch. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 590–601, 2023.

[37] Siyuan Dong, Zhuochen Fan, Tianyu Bai, Tong Yang, Hanyu Xue, Peiqing Chen, and Yuhan Wu. M4: A framework for per-flow quantile estimation. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 4787–4800. IEEE, 2024.

[38] Zhuochen Fan, Yalun Cai, Siyuan Dong, Qiuheng Yin, Tianyu Bai, Hanyu Xue, Peiqing Chen, Yuhan Wu, Tong Yang, and Bin Cui. Per-flow quantile estimation using m4 framework. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[39] The source code and technical report related to KLL-Polymer. https://github.com/KLL-Polymer/KLL-Polymer-code.

[40] J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.

[41] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. *ACM SIGMOD Record*, 28(2):251–262, 1999.

[42] Graham Cormode and Pavel Veselỳ. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*, pages 81–93, 2020.

[43] David Felber and Rafail Ostrovsky. A randomized online quantile summary in $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ words. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, page 775, 2015.

[44] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009.

[45] Charles Masson, Jee E Rim, and Homin K Lee. Ddsketch: A fast and fully-mergeable quantile sketch with relative-error guarantees. *arXiv preprint arXiv:1908.10693*, 2019.

[46] Ted Dunning and Otmar Ertl. Computing extremely accurate quantiles using t-digests. *arXiv preprint arXiv:1902.04023*, 2019.

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

[48] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[49] Graham Cormode, Zohar Karnin, Edo Liberty, Justin Thaler, and Pavel Veselỳ. Relative error streaming quantiles. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 96–108, 2021.

[50] Graham Cormode, Abhinav Mishra, Joseph Ross, and Pavel Veselỳ. Theory meets practice at the median: A worst case comparison of relative error quantile algorithms. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2722–2731, 2021.

[51] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International conference on database theory*, pages 398–412. Springer, 2005.
[52] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.
[53] Fuheng Zhao, Sujaya Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. Kll± approximate quantile sketches over dynamic datasets. *Proceedings of the VLDB Endowment*, 14(7):1215–1227, 2021.
[54] Ziling Chen and Shaoxu Song. Randomized sketches for quantile in lsm-tree based store. *Proceedings of the ACM on Management of Data*, 3(1):1–26, 2025.
[55] Qilong Shi, Wei Zhou, Yizhuo Zheng, Xinye Xu, Ting Yang, Yuanyuan Zhang, Long Yao, Yangyang Wang, and Mingwei Xu. Cooled-kll: Enhancing quantile estimation by filtering hot item. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 2562–2573, 2025.
[56] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.
[57] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):1–28, 2013.
[58] The source code of Bob Hash. http://burtleburtle.net/bob/hash/evahash.html.
[59] CAIDA Anonymized Internet Traces 2018 Dataset. https://www.caida.org/catalog/datasets/overview/.
[60] MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi/.
[61] Facebook RocksDB. http://rocksdb.org/.

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026

A Comparison of Per-Key Algorithms

We show the properties of different per-key algorithms in Table 3, including their space complexity, whether the space complexity depends on data distribution, whether the algorithm is comparison-based, and mergeability. Note that HistSketch, SketchPolymer, and M4 include stages for filtering infrequent keys, so their space complexity depends on the actual data distribution.

Algorithm	Space Complexity	Distribution-dependent	Comparison-based	Mergeability
HistSketch [33]	Not explicitly stated	Yes	No	Yes
SQUAD [34, 35]	$O\left(\frac{1}{\theta\varepsilon^{1.5}}\log\frac{1}{\delta}\right)$	No	Yes	Yes
SketchPolymer [36]	$O\left(\frac{1}{\theta\varepsilon}\log\frac{1}{\delta}\right)$	Yes	No	Yes
M4 [37, 38]	$mw\log\frac{2}{\delta}\times$ Space(META)	Yes	No	Yes
KLL-Polymer (Ours)	$O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$	No	Yes	Yes
KLL-Polymer-DC (Ours)	At most $O\left(\frac{1}{\theta\varepsilon}\log^2\log\frac{1}{\delta}\right)$	Yes	Yes	No

Table 3. Comparison of existing per-key algorithms.

B Details of Datasets

The details of each dataset are shown in Table 4.

Type	Dataset	# Items	# Keys	θ	# Heavy Hitters	Key Distribution	Value Distribution
Real	CAIDA	20,000,000	9,417,590	10^{-4}	298	—	—
	MAWI	10,000,000	2,173,066	10^{-3}	45	—	—
Synthetic		10,000,000	1000	10^{-3}	517	Uniform	Gamma
		10,000,000	1000	10^{-3}	497	Uniform	Lognormal
		10,000,000	100,000	10^{-3}	52	Zipfian, $\alpha = 1.5$	Gamma
		10,000,000	100,000	10^{-3}	52	Zipfian, $\alpha = 1.5$	Lognormal

Table 4. Details of datasets.

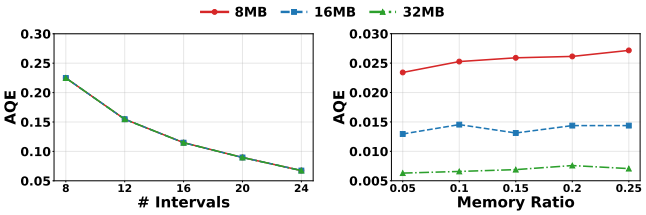
C Parameter Settings for Prior Work

Here, we introduce the parameter settings for HistSketch, SQUAD, SketchPolymer and M4.

HistSketch: HistSketch divides values into a fixed number of intervals in advance, and employs a two-stage data structure to handle heavy and light intervals separately. We construct these intervals such that each contains approximately the same number of values.[7] We conduct experiments on this interval quantity parameter, and Figure 17(a) demonstrates that a larger number of intervals leads to a lower error, while the impact of the memory budget is not significant. However, partitioning the data into more intervals with equal counts relies on more precise prior information about the underlying distribution, which is difficult in practical streaming scenarios. Therefore, referring to the default setting in the original HistSketch paper, we fix the number of intervals to 16. The total memory is evenly divided between the two stages.

[7]Note that constructing intervals with equal counts requires prior knowledge of the value distribution, which is not available in streaming models. We nevertheless adopt this setup to allow HistSketch to run in our evaluation.

SQUAD: SQUAD states that keeping $O\left(\frac{1}{\theta\epsilon^{1.5}}\log\frac{1}{\delta}\right)$ items via Reservoir Sampling and $O\left(\frac{1}{\theta\epsilon^{0.5}}\right)$ buckets via the Space Saving algorithm suffices to solve the per-key estimation. However, SQUAD does not explicitly specify how memory should be allocated when the total space is fixed. To address this, we conduct experiments by varying the ratio of memory allocated to Reservoir Sampling relative to the total space. As illustrated in Figure 17(b), allocating more space to the Space Saving component generally yields better experimental accuracy. Nevertheless, as long as the majority of the total space is dedicated to Space Saving, the estimation error exhibits only minor fluctuations. Therefore, we allocate 10% of the memory for Reservoir Sampling and the remaining 90% for Space Saving. Within each Space Saving bucket, we employ a KLL sketch for quantile estimation, using the same parameters $c = \frac{2}{3}$ and $s = 3$ for fairness.



(a) HistSketch (b) SQUAD

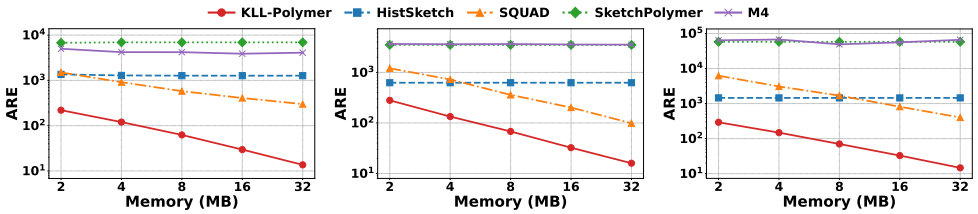
Fig. 17. Effects of parameters for HistSketch and SQUAD.

SketchPolymer: SketchPolymer uses a four-stage data structure, with Stage 1 for filtering infrequent keys, and Stages 2–4 for quantile approximation. According to its default settings, we allocate 5%, 30%, 50%, and 15% of the total memory to Stages 1 through 4, respectively. Since each stage of SketchPolymer applies a Count-Min Sketch or a Bloom Filter, we set the number of hash functions to 3, 5, 5, and 3 for the four stages. The threshold for filtering infrequent keys is set to 50, *i.e.* keys with frequency exceeding 50 will enter Stages 2–4. Finally, as SketchPolymer divides values into intervals logarithmically, we set the logarithm base to 1.5, *i.e.* values within $[1.5^n, 1.5^{n+1})$ fall in the same interval.

M4: M4 also applies a data structure with four layers, where each bucket in each layer applies a single-key algorithm META. Based on the recommended configuration of M4, Layers 1, 2, 3, and 4 maintain keys with frequencies within $[0, 4)$, $[4, 256)$, $[256, 65536)$ and $[65536, +\infty)$ respectively. We hash each item into 3 buckets in each layer, and allocate 3% memory for Layer 1, 60% memory for Layer 2, 35% memory for Layer 3, and 2% memory for Layer 4. For the META algorithm, we adopt ReqSketch.

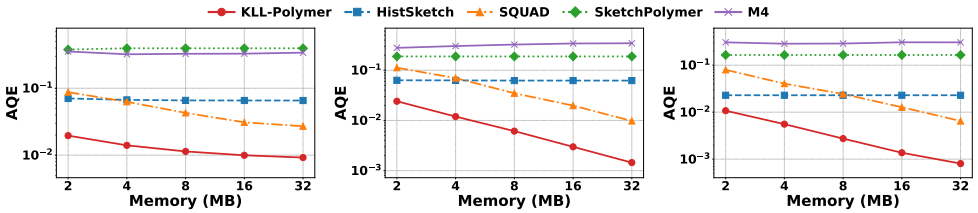
D Extended Experiments in Section 5.2

Here, we present the experimental results of KLL-Polymer and prior work on the MAWI, Uniform-Lognormal, and Zipfian-Gamma datasets.



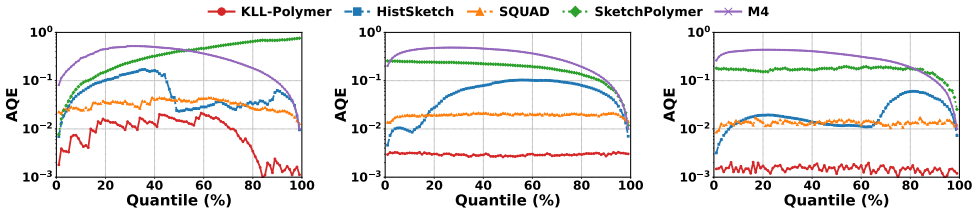
(a) MAWI (b) Uniform-Lognormal (c) Zipfian-Gamma

Fig. 18. Average Rank Error (ARE) on different datasets.



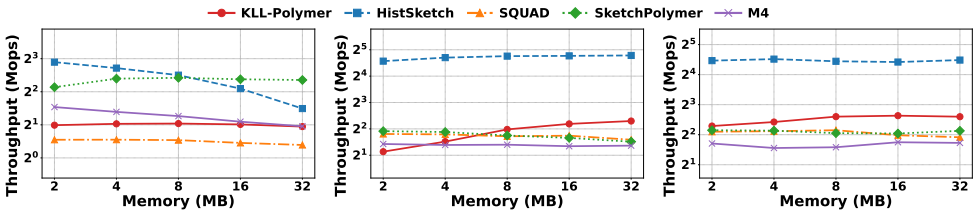
(a) MAWI (b) Uniform-Lognormal (c) Zipfian-Gamma

Fig. 19. Average Quantile Error (AQE) on different datasets.



(a) MAWI (b) Uniform-Lognormal (c) Zipfian-Gamma

Fig. 20. Average Quantile Error (AQE) on different quantiles.



(a) MAWI (b) Uniform-Lognormal (c) Zipfian-Gamma

Fig. 21. Throughput on different datasets.

E Extended Experiments on Randomness

Since KLL-Polymer relies on randomized odd-even sampling to execute compaction operations, we evaluate its empirical stability and variance in this section. Specifically, we repeat each set of experiments 10 times using different random seeds under a fixed 8MB memory budget. We report the key statistical characteristics of the estimation errors—including the maximum, minimum, mean, and standard deviation. The detailed results are presented in Table 5, which clearly demonstrate that the variance of KLL-Polymer is virtually negligible, ensuring highly robust performance in practice.

Datasets	Min	Max	Mean	Std
CAIDA	106.2	118	113.8	3.04
MAWI	55.21	64.45	59.23	3.35
Uniform-Gamma	65.55	68.63	67.16	0.93
Uniform-Lognormal	66.64	69.53	67.78	0.94
Zipfian-Gamma	62.49	77.11	67.94	4.77
Zipfian-Lognormal	63.06	72.57	67.99	2.79

(a) Average Rank Error (ARE)

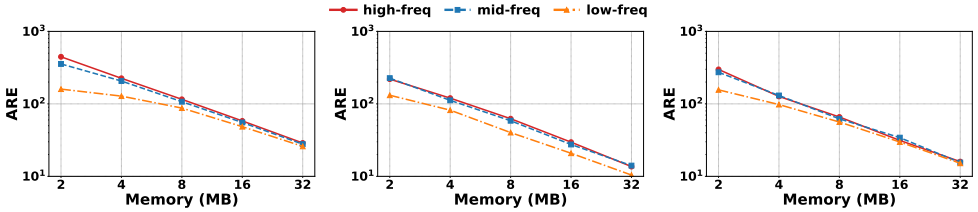
Datasets	Min	Max	Mean	Std
CAIDA	0.0385	0.0412	0.04	0.001
MAWI	0.0111	0.0116	0.0114	0.00017
Uniform-Gamma	0.00603	0.00619	0.00612	0.00006
Uniform-Lognormal	0.00597	0.00618	0.0061	0.00006
Zipfian-Gamma	0.00246	0.00285	0.00263	0.00013
Zipfian-Lognormal	0.00115	0.00131	0.00124	0.00005

(b) Average Quantile Error (AQE)

Table 5. Statistical characteristics of estimation errors on different datasets.

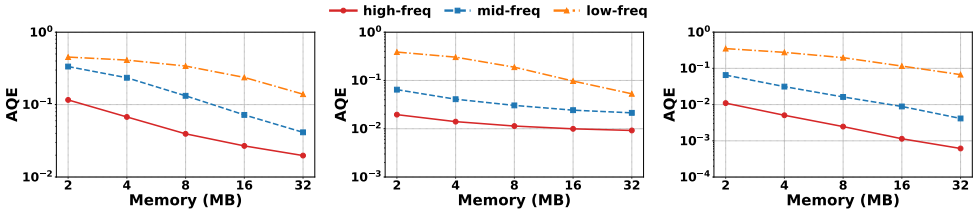
F Experiments on Non-heavy-hitter Keys

To empirically validate our theoretical remarks on non-heavy-hitters, we evaluate the performance of KLL-Polymer across distinct keys with varying frequencies. We restrict our query set to keys with frequency $f_k \geq 100$, and categorize these keys into three non-overlapping tiers: high-frequency (keys with $f_k \geq \theta N$), mid-frequency (keys with $0.2\theta N \leq f_k < \theta N$), and low-frequency ($100 \leq f_k < 0.2\theta N$). According to Figure 22, non-heavy-hitter keys achieve comparable or even superior estimation errors in rank queries compared to heavy hitters. Their items are involved in fewer compaction operations due to lower frequency, which leads to a lower absolute rank error. However, Figure 23 reveals a starkly different reality for quantile queries. Non-heavy-hitter keys suffer from significantly higher AQE than heavy hitters, frequently exceeding 0.1 under tight memory budgets. Such a massive error margin fundamentally distorts the internal distribution, rendering these quantile estimates practically meaningless. This empirical evidence corroborates our theoretical remark: while KLL-Polymer supports queries for all keys, the practical utility of quantile estimations for extremely infrequent keys is severely limited.



(a) CAIDA (b) MAWI (c) Zipfian-Lognormal

Fig. 22. Average Rank Error (ARE) across different frequency tiers.



(a) CAIDA (b) MAWI (c) Zipfian-Lognormal

Fig. 23. Average Quantile Error (AQE) across different frequency tiers.