AeroSketch: Near-Optimal Time Matrix Sketch Framework for Persistent, Sliding Window, and Distributed Streams

HANYAN YIN, Renmin University of China, China
DONGXIE WEN, Renmin University of China, China
JIAJUN LI, Renmin University of China, China
ZHEWEI WEI*, Renmin University of China, China
XIAO ZHANG, Renmin University of China, China
PENG ZHAO, Nanjing University, China
ZHI-HUA ZHOU, Nanjing University, China

Many real-world matrix datasets arrive as high-throughput vector streams, making it impractical to store or process them in their entirety. To enable real-time analytics under limited computational, memory, and communication resources, matrix sketching techniques have been developed over recent decades to provide compact approximations of such streaming data. Some algorithms have achieved optimal space and communication complexity. However, these approaches often require frequent time-consuming matrix factorization operations. In particular, under tight approximation error bounds, each matrix factorization computation incurs cubic time complexity, thereby limiting their update efficiency.

In this paper, we introduce AeroSketch, a novel matrix sketching framework that leverages recent advances in randomized numerical linear algebra (RandNLA). AeroSketch achieves optimal communication and space costs while delivering near-optimal update time complexity (within logarithmic factors) across persistent, sliding window, and distributed streaming scenarios. Extensive experiments on both synthetic and real-world datasets demonstrate that AeroSketch consistently outperforms state-of-the-art methods in update throughput. In particular, under tight approximation error constraints, AeroSketch reduces the cubic time complexity to the quadratic level. Meanwhile, it maintains comparable approximation quality while retaining optimal communication and space costs.

CCS Concepts: • **Theory of computation → Sketching and sampling**; *Data compression*; Vector / streaming algorithms.

Additional Key Words and Phrases: Streaming Data, Optimal Time Complexity, Matrix Sketch, Persistent Sketch, Sliding Window, Distributed Sketch

ACM Reference Format:
Hanyan Yin, Dongxie Wen, Jiajun Li, Zhewei Wei, Xiao Zhang, Peng Zhao, and Zhi-Hua Zhou. 2026. AeroSketch: Near-Optimal Time Matrix Sketch Framework for Persistent, Sliding Window, and Distributed Streams. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 8 (February 2026), 26 pages. https://doi.org/10.1145/3786622

*Zhewei Wei is the corresponding author. The work was partially done at Gaoling School of Artificial Intelligence, Beijing Key Laboratory of Research on Large Models and Intelligent Governance, Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE, and Pazhou Laboratory (Huangpu), Guangzhou, Guangdong 510555, China.

Authors' Contact Information: Hanyan Yin, yinhanyan@ruc.edu.cn, Renmin University of China, Beijing, China; Dongxie Wen, 2019202221@ruc.edu.cn, Renmin University of China, Beijing, China; Jiajun Li, 2015201613@ruc.edu.cn, Renmin University of China, Beijing, China; Zhewei Wei, zhewei@ruc.edu.cn, Renmin University of China, Beijing, China; Xiao Zhang, zhangx89@ruc.edu.cn, Renmin University of China, Beijing, China; Peng Zhao, zhaop@lamda.nju.edu.cn, Nanjing University, Nanjing, Jiangsu, China; Zhi-Hua Zhou, zhouzh@lamda.nju.edu.cn, Nanjing University, Nanjing, Jiangsu, China.

ACM 2836-6573/2026/2-ART8
https://doi.org/10.1145/3786622

1 Introduction

Many types of real-world data—such as computer network traffic [16, 39], online machine learning [3, 4, 18, 36, 37], and sensor data [2, 41]—are continuously generated as vast, high-velocity streams of vectors [28, 34, 46]. Storing even a reasonable portion of these streams in memory is impractical. Therefore, it is necessary to design online algorithms that process one vector at a time and immediately discard it, enabling real-time processing with reduced memory usage at the cost of some precision. Such compact representations are known as *matrix sketching* [5, 7, 15, 20, 38].

Table 1. Given the dimension d of each row vector (for tracking approximate matrix multiplication over sliding window (Problem 4), there are two vector streams with dimensions d_x and d_y, respectively), the upper bound on the relative covariance error ε, the norm of each row $\|\boldsymbol{a}_i\|_2^2 \in [1, R]$, and the number of sites m (in distributed scenarios), this table presents an overview of the communication/space and amortized update time complexities of algorithms addressing a wide range of matrix sketching problems. Here, $\delta \in (0, 0.01)$ denotes the upper bound on failure probability; that is, each listed complexity bound holds for an algorithm that returns a matrix sketch satisfying the stated error guarantees with probability at least $1 - \delta$. ✓ indicates that the asymptotic complexity is optimal; – indicates that it is not optimal; and ✓* denotes near-optimal complexity (up to a logarithmic factor).

Problem Scenarios		Methods	Space / Communication		Amortized Update Time Complexity	
Full Stream Matrix Sketch (Problem 1)		Frequent Directions [8, 20]	$O\left(\frac{d}{\varepsilon}\right)$	✓	$O\left(\frac{d}{\varepsilon}\right)$	✓
Tracking Matrix Sketch Problem (Scenario 1)	Tracking Matrix Sketch over Sliding Window (Problem 2)	Fast-DS-FD [44]	$O\left(\frac{d}{\varepsilon}\log R\right)$	✓	$O\left(\left(\frac{d}{\varepsilon}+\frac{1}{\varepsilon^3}\right)\log R\right)$	–
		[Theorem 3.1]	$O\left(\frac{d}{\varepsilon}\log R\right)$	✓	$O\left(\frac{d}{\varepsilon}\log d\log R\right)$	✓*
		[Corollary 3.2]	$O\left(\frac{d}{\varepsilon}\log R\right)$	✓	$O\left(\frac{d}{\varepsilon}\log d\log R\log^2\frac{1}{\delta}\right)$	✓*
	Tracking Persistent Matrix Sketch (ATTP, Problem 3)	ATTP-FD [30]	$O\left(\frac{d}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$	✓	$O\left(\frac{d}{\varepsilon^2}\right)$	–
		[Theorem 4.1]	$O\left(\frac{d}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$	✓	$O\left(\frac{d}{\varepsilon}\log d\log^2\frac{1}{\delta}\right)$	✓*
	Tracking Approximate Matrix Multiplication over Sliding Window (Problem 4)	ML-SO-COD [40] hDS-COD [42]	$O\left(\frac{d_x+d_y}{\varepsilon}\log R\right)$	✓	$O\left(\left(\frac{d_x+d_y}{\varepsilon}+\frac{1}{\varepsilon^3}\right)\log R\right)$	–
		[Theorem 4.2]	$O\left(\frac{d_x+d_y}{\varepsilon}\log R\right)$	✓	$O\left(\frac{d_x+d_y}{\varepsilon}\log d\log R\log^2\frac{1}{\delta}\right)$	✓*
Tracking Distributed Matrix Sketch Problem (Scenario 2)	Tracking Matrix Sketch over Distributed Sites (Problem 5)	P2 [10]	$O\left(\frac{md}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$	✓	$O\left(\frac{d}{\varepsilon^2}\right)$	–
		[Theorem 4.3]	$O\left(\frac{md}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$	✓	$O\left(\frac{d}{\varepsilon}\log d\log^2\frac{1}{\delta}\right)$	✓*
	Tracking Matrix Sketch over Distributed Sliding Window (Problem 6)	DA2 [45]	$O\left(\frac{md}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$	✓	$O\left(\frac{d}{\varepsilon^2}\right)$	–
		[Theorem 4.4]	$O\left(\frac{md}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$	✓	$O\left(\frac{d}{\varepsilon}\log d\log^2\frac{1}{\delta}\right)$	✓*

We observe that a range of matrix sketching problems over vector streams—including tracking persistent matrix sketches [30], matrix sketches over sliding windows [35, 44], distributed matrix sketches [10, 45], and approximate matrix multiplication [40, 42]—can be reduced to two fundamental tasks: *tracking matrix sketch problem with limited memory* (Scenario 1) and *tracking distributed matrix sketch problem over bandwidth-constrained channels* (Scenario 2), as categorized in Table 1. The primary optimization metrics of matrix sketch algorithms are space cost (for Scenario 1), communication cost (for Scenario 2), and time cost (for both Scenarios 1 and 2).

Extensive research has been dedicated to developing efficient matrix sketch algorithms for various problem settings [8–10, 20, 30, 35, 40, 42, 44, 45], as well as theoretically analyzing the relationships

among approximation quality, space/communication cost, and time complexity [8, 13, 14, 21, 44]. [13, 14] proved a lower bound on the communication cost for the distributed deterministic matrix sketch problem. In other words, any deterministic matrix sketching algorithm that solves the problem under Scenario 2 must incur a communication cost of at least a certain number of bits. The corresponding lower bounds for the space complexities under Scenario 1 have also been proved by other researchers [8, 40, 42, 44]. In addition to the theoretical results, numerous specific matrix sketch algorithms have been proposed. Some of them have the same asymptotic communication/space complexity as the theoretical lower bound, indicating that the sketch algorithms have achieved optimal communication/space costs, as listed in Table 1 and marked by ✓.

Despite their superiority in communication/space complexity, these algorithms often suffer from high computational costs, especially under tight approximation error bounds and over high-dimensional vector streams of dimension d. Their asymptotic time complexities are suboptimal compared with the time lower bound proved by Huang [12] (marked by — in Table 1). This is mainly because they continuously perform time-consuming matrix factorizations to monitor changes in the stream and update the matrix sketch. In particular, under tight approximation error bounds, each matrix factorization incurs a cubic time complexity of $O(d^3)$, which is prohibitive for high-dimensional vector streams. As a result, the prior communication- and space-optimal algorithms are far from optimal in terms of computational complexity. This underperformance naturally raises the question: *Can we design matrix sketching algorithms that are simultaneously optimal in communication, space, and time complexity?*

Fortunately, we find that some more efficient RandNLA (randomized numerical linear algebra) operators, including Power Iteration and Simultaneous Iteration, can replace time-consuming deterministic matrix factorization operators [11, 22, 23, 26, 32]. Although substituting with RandNLA operators may seem straightforward, there is a cost: RandNLA operators introduce probabilistic errors, but deterministic matrix factorization operators do not. As a result, key steps in prior algorithms must also be adapted. Moreover, randomness inherent in RandNLA makes it difficult to determine appropriate parameter settings. It is also challenging to rigorously analyze how probabilistic errors affect the final matrix sketch's accuracy, as well as the communication, space, and time complexity. Another obstacle is that RandNLA operators may not directly meet requirements. For example, we need to identify singular values and vectors that exceed a given threshold θ, but Simultaneous Iteration only supports computing the top-k singular components for a specified k. However, we do not know in advance how many of the top-k singular values exceed the threshold θ. This adds further difficulty to both algorithm design and theoretical complexity analysis.

In this work, we overcome these obstacles through proper algorithm design and parameter selection. The stochastic error from RandNLA operators can be analytically regulated, while maintaining optimal space/communication complexity on par with the best baseline methods. The amortized time complexity per update also approaches near-optimality (within a logarithmic factor). The new generalized matrix sketching framework is termed AeroSketch, which accelerates a wide range of tracking matrix sketching algorithms to near-optimal update time complexity (marked as ✓* in Table 1) along with optimal communication/space complexity. The specific contributions of this paper are outlined below:

- **Novel Framework:** For a wide range of tracking matrix sketching problems listed in Table 1, we design a new generalized matrix sketching framework, termed AeroSketch, which improves the time efficiency of the previous state-of-the-art methods, while maintaining their optimal communication and space overhead. In particular, under tight approximation error constraints ε, baseline algorithms degrade to $O(d^3)$ time complexity while our AeroSketch framework achieves $O(d^2 \log d)$ time complexity per update.

- **Rigorous Theoretical Analysis:** We provide a rigorous end-to-end proof of AeroSketch, demonstrating its competitive approximation quality, communication and space complexities, and superior update time complexities compared to state-of-the-art methods, especially under tight approximation error constraints ε. To the best of our knowledge, AeroSketch is the first implementable algorithm with theoretical guarantees of explicit probabilistic error bounds, near-optimal update time complexity, and optimal space/communication complexity.
- **Extensive Experiments:** We implement the AeroSketch framework and conduct comprehensive experiments to verify its superiority over state-of-the-art algorithms, particularly in terms of empirical approximation error, communication/space cost, and time consumption. Experimental results show that the trade-off between error and computational cost for the AeroSketch-optimized algorithm is more favorable than that for the best baseline algorithms on both synthetic and real-world datasets. The communication/space cost of our framework is also competitive. Furthermore, optimizing time cost becomes increasingly significant as the upper bound on covariance relative error ε tightens and the dimensionality of the vectors d increases substantially. These experimental findings strongly align with our theoretical analyses.

2 Preliminaries and Related Work

This section introduces problem definitions and relevant tools, including Power Iteration, Simultaneous Iteration, and Frequent Directions, on which our AeroSketch framework is based.

2.1 Problem Definition: Matrix Sketching

We now provide formal definitions of the different variants of matrix sketching problems listed in Table 1. A stream $\boldsymbol{A}$ consists of row vectors $\boldsymbol{a}_i \in \mathbb{R}^{1\times d}$, where the norm of each row satisfies $\|\boldsymbol{a}_i\|_2^2 \in [1, R]$. Given two time values $s < t$, define $\boldsymbol{A}_{s,t} \in \mathbb{R}^{(t-s)\times d}$ as the portion of the stream that arrived during the time interval $(s, t]$. Let 0 denote a time point before any elements of the stream have arrived, and T denote the current time. We further define the specific stream subsets $\boldsymbol{A}_t = \boldsymbol{A}_{0,t}$ and $\boldsymbol{A}_{-t} = \boldsymbol{A}_{t,T}$. The sizes of both the sketching state and the output sketch matrix are significantly smaller than the matrix to be approximated. In distributed settings, the communication cost between the m sites and the coordinator, as well as the size of the output matrix, is significantly smaller than that of the matrix to be approximated.

2.1.1 Full Stream Matrix Sketch. The goal of the *full stream matrix sketch* problem is to maintain a matrix sketch of a stream from its beginning up to the current time T. The formal definition is [8, 20]:

Problem 1 (Full Stream Matrix Sketching)**.** Given an error parameter ε, at any current time T, we must maintain a matrix sketch $\boldsymbol{B}_T$ such that $\boldsymbol{B}_T$ approximates the matrix $\boldsymbol{A}_T$. The approximation quality is assured by the *covariance error* bound, defined as follows:

$$\textbf{cov-error}(\boldsymbol{A}_T, \boldsymbol{B}_T) = \|\boldsymbol{A}_T^\top \boldsymbol{A}_T - \boldsymbol{B}_T^\top \boldsymbol{B}_T\|_2 \le O(1)\varepsilon \|\boldsymbol{A}_T\|_F^2.$$

2.1.2 Matrix Sketching over Sliding Window. In real-world scenarios, the focus often lies on the most recently arrived elements rather than outdated items within data streams. Datar et al. [6] considered the problem of maintaining aggregates and statistics from the most recent portion of a data stream and referred to such a model as the *sliding window model*. The *matrix sketching over sliding window* problem is formally stated as follows [35, 44]:

Problem 2 (Matrix Sketching over Sliding Window)**.** Given an error parameter ε and a window size N, the goal is to maintain a matrix sketch $\boldsymbol{B}_\mathrm{W}$ such that, at the current time T, $\boldsymbol{B}_\mathrm{W}$ approximates the matrix $\boldsymbol{A}_\mathrm{W} = \boldsymbol{A}_{T-N,T} \in \mathbb{R}^{N\times d}$. The approximation quality is assured by the *covariance error*

bound:

$$\textbf{cov-error}(\boldsymbol{A}_{\mathrm{W}}, \boldsymbol{B}_{\mathrm{W}}) = \|\boldsymbol{A}_{\mathrm{W}}^{\top}\boldsymbol{A}_{\mathrm{W}} - \boldsymbol{B}_{\mathrm{W}}^{\top}\boldsymbol{B}_{\mathrm{W}}\|_2 \leq O(1)\varepsilon\|\boldsymbol{A}_{\mathrm{W}}\|_F^2.$$

Remark. *Besides the space-optimal Fast-DS-FD algorithm proposed in [44] that is listed in Table 1, a time-optimized version called* ***probabilistic Fast-DS-FD*** *is also mentioned in [44], claiming that the per-update time complexity is $O(d\ell)$. By setting $\ell \leftarrow \min\left(\lceil \frac{1}{\varepsilon} \rceil, d\right)$, the time complexity becomes $O\left(\frac{d}{\varepsilon}\right)$. The authors position probabilistic Fast-DS-FD as a potential direction for future work, as a fully detailed implementation is not provided. Important parameters are not defined explicitly, such as how to set the number of iterations for the RandNLA algorithm to meet the error bound. Therefore, we did not list probabilistic Fast-DS-FD in Table 1 as a competitor to our AeroSketch framework. Additionally, we found that Lemma 1 in [44], which involves the key operation of dumping and restoring snapshots and the associated correctness proof, holds only under the premise that $\boldsymbol{v}_j$ is an exact singular vector, which is true for Fast-DS-FD. However, the probabilistic randomized algorithm used by probabilistic Fast-DS-FD to optimize time can obtain only an approximate estimate of $\boldsymbol{v}_j$. Therefore, directly replacing the SVD in Fast-DS-FD with RandNLA will introduce cumulative restoring norm errors, as shown in Figure 1. Designing the dumping and restoring snapshots operation and correctness proof under this premise is far from straightforward. In this paper, our AeroSketch framework will overcome these limitations.*

2.1.3 (At-the-time) Persistent Matrix Sketching. In this paper, we focus on the ATTP (*at-the-time persistent*) sketch task proposed by Shi et al. [30], which requires supporting queries on arbitrary historical versions of the full stream matrix sketch. The formal definition is as follows:

Problem 3 (ATTP Sketch)**.** Given an error parameter ε, we maintain a matrix sketch such that, at the current time T, it can return an approximation $\boldsymbol{B}_t$ for any matrix $\boldsymbol{A}_t = \boldsymbol{A}_{0,t} \in \mathbb{R}^{t\times d}$ with $t \leq T$. The approximation quality is assured by the *covariance error* bound:

$$\forall t < T, \quad \textbf{cov-error}(\boldsymbol{A}_t, \boldsymbol{B}_t) = \|\boldsymbol{A}_t^{\top}\boldsymbol{A}_t - \boldsymbol{B}_t^{\top}\boldsymbol{B}_t\|_2 \leq O(1)\varepsilon\|\boldsymbol{A}_t\|_F^2.$$

2.1.4 Approximate Matrix Multiplication over Sliding Window. We follow the problem definition of *approximate matrix multiplication over sliding window* as in [40, 42, 43]:

Problem 4 (Approximate Matrix Multiplication over Sliding Window)**.** Let $\{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t\geq 0}$ be sequences of two vector streams, where for each time t, we have $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$ and $\boldsymbol{y}_t \in \mathbb{R}^{d_y}$. For a fixed window size N and for any time $T \geq N$, define the sliding window matrices:

$$\boldsymbol{X}_{\mathrm{W}} = \begin{bmatrix} \boldsymbol{x}_{T-N+1} & \boldsymbol{x}_{T-N+2} & \cdots & \boldsymbol{x}_T \end{bmatrix} \in \mathbb{R}^{d_x\times N},$$
$$\boldsymbol{Y}_{\mathrm{W}} = \begin{bmatrix} \boldsymbol{y}_{T-N+1} & \boldsymbol{y}_{T-N+2} & \cdots & \boldsymbol{y}_T \end{bmatrix} \in \mathbb{R}^{d_y\times N}.$$

The goal is to yield, at every time $T \geq N$, matrices $\boldsymbol{A}_{\mathrm{W}} \in \mathbb{R}^{d_x\times\ell}$ and $\boldsymbol{B}_{\mathrm{W}} \in \mathbb{R}^{d_y\times\ell}$ satisfying:

$$\left\|\boldsymbol{X}_{\mathrm{W}}\boldsymbol{Y}_{\mathrm{W}}^{\top} - \boldsymbol{A}_{\mathrm{W}}\boldsymbol{B}_{\mathrm{W}}^{\top}\right\|_2 \leq O(1)\epsilon\,\|\boldsymbol{X}_{\mathrm{W}}\|_F\,\|\boldsymbol{Y}_{\mathrm{W}}\|_F.$$

Remark. *Regarding Problem 4, we list two space-optimal baseline algorithms in Table 1: ML-SO-COD proposed by Xian et al. [40] and hDS-COD proposed by Yao et al. [42]. Among them, [40] explicitly states in Theorem 3 that the per-update time complexity of ML-SO-COD is $O\left(\left(\left(d_x + d_y\right)\ell + \ell^3\right)\log R\right)$. Meanwhile,we note that the proof of Theorem 1 in [42] writes: "the amortized time complexity for a single update step in Algorithm 4 (hDS-COD) is $O((d_x + d_y)\ell + \ell^3)$. " (the last paragraph of Appendix C in [40]). As ε converges to 0, according to Theorem 1 in [40], ℓ is set to $\ell = 1/\varepsilon$, consequently it is often the case that $d_x + d_y = O(\ell^2)$ or even $d_x + d_y = O(\ell)$. Under such conditions, substituting this into complexity $O((d_x + d_y)\ell + \ell^3)$ means the ℓ^3 term can no longer be neglected, causing the complexity to degrade to $O((d_x + d_y)^3)$, matching the per-update time complexity of ML-SO-COD under the same condition of $d_x + d_y = O(\ell)$.*

Additionally, Section 4.3 of [42] proposes Adaptive DS-COD (aDS-COD) in Algorithm 6, which optimizes both time and space complexity based on hDS-COD. Meanwhile, it is explicitly pointed out that this algorithm is only suitable for streams that "have a large R but little fluctuation within the window", and not for the more strictly defined version in Problem 4, which requires handling arbitrary fluctuations within the window. Thus, for fairness in comparison, we do not include aDS-COD in Table 1 as a competitor to our AeroSketch framework. Nonetheless, we find that our AeroSketch framework can replace the DS-COD component in aDS-COD, further optimizing its time complexity under conditions where ε is small. We will demonstrate the experimental results of this optimization in our technical report [1].

2.1.5 Tracking Distributed Matrix Sketch. In many scenarios, such as distributed databases, wireless sensor networks, and cloud computing, there are multiple distributed input streams. Each stream is observed by one of the m distributed sites. Suppose we have a distributed data stream $A = \{(\boldsymbol{a}_i, t_i, S_i) \mid i = 1, 2, \dots\}$, where each item $\boldsymbol{a}_i \in \mathbb{R}^{1\times d}$ is a d-dimensional vector with timestamp t_i arriving at site $S_i \in \{1, 2, \dots, m\}$. We adopt the definition of *tracking distributed matrix sketch* given by Ghashami et al. [10]:

Problem 5 (Tracking Distributed Matrix Sketch)**.** At any current time T, define the matrix $\boldsymbol{A}_T$ as the collection of all rows $\boldsymbol{a}_i$ across all m sites with timestamps in the interval $(0, T]$. The goal is to maintain, at a central coordinator, a compact sketch matrix $\boldsymbol{B}_T$ such that it approximates $\boldsymbol{A}_T$, with the guarantee of the covariance error bound:

$$\textbf{cov-error}(\boldsymbol{A}_T, \boldsymbol{B}_T) = \|\boldsymbol{A}_T^\top \boldsymbol{A}_T - \boldsymbol{B}_T^\top \boldsymbol{B}_T\|_2 \le O(1)\varepsilon \|\boldsymbol{A}_T\|_F^2.$$

In distributed settings, attention is also often focused on recent elements, known as the problem of *tracking distributed matrix sketch over sliding window*. The definition given by Zhang et al. [45] is:

Problem 6 (Tracking Distributed Matrix Sketch over Sliding Window)**.** Given a window size N, define $\boldsymbol{A}_{\mathrm{W}}$ to consist of all vectors $\boldsymbol{a}_i$ across all m sites with timestamps in the interval $(T - N, T]$. Maintain a compact sketch matrix $\boldsymbol{B}_{\mathrm{W}}$ at a central coordinator such that it approximates $\boldsymbol{A}_{\mathrm{W}}$ with bounded covariance error:

$$\textbf{cov-error}(\boldsymbol{A}_{\mathrm{W}}, \boldsymbol{B}_{\mathrm{W}}) = \|\boldsymbol{A}_{\mathrm{W}}^\top \boldsymbol{A}_{\mathrm{W}} - \boldsymbol{B}_{\mathrm{W}}^\top \boldsymbol{B}_{\mathrm{W}}\|_2 \le O(1)\varepsilon \|\boldsymbol{A}_{\mathrm{W}}\|_F^2.$$

Remark. *In Table 1, the baseline algorithm we list for solving Problem 6 is DA2, proposed by Zhang et al. [45]. Although the update time complexity of DA2 is not explicitly given in [45], since each update requires performing an SVD on a matrix of size $d \times O(1/\varepsilon)$, the update time complexity of DA2 should be dominated by this operation, with a time cost of $O(d/\varepsilon^2)$. When $\varepsilon < O(1/d)$, the time complexity degrades to $O(d^3)$. An analysis in the experimental section of the original paper also confirms this: "This is because DA2 needs to compute matrix factorizations periodically, which leads to running time quadratic or even cubic in d." (last paragraph of Sec IV.B in [45]).*

2.2 RandNLA Operators

2.2.1 Power Iteration. **Power Iteration** (also known as the **Power Method**) is a classic algorithm for computing an approximate largest singular value [17, 31]. Given a matrix $\boldsymbol{A}$, the algorithm produces an estimate $\hat{\sigma}_1^2$ of the squared largest singular value of $\boldsymbol{A}$, along with an estimated corresponding top singular vector $\hat{\boldsymbol{v}}_1$. The pseudocode is shown in Algorithm 1.

A textbook theorem about the convergence rate of Power Iteration states the following: if the initial vector $\boldsymbol{x}^{(0)}$ in line 1 of Algorithm 1 is not orthogonal to the top singular vector, then after k iterations the gap between the estimated squared singular value $\hat{\sigma}_1^2$ and the true squared top

Algorithm 1: Power Iteration Algorithm

Input: $\boldsymbol{A} \in \mathbb{R}^{d\times\ell}$: the matrix for which the approximate largest singular value (squared) and corresponding singular vector are to be computed; k: the number of iterations.

Output: $\hat{\sigma}_1^2$: the squared largest singular value of $\boldsymbol{A}$; $\boldsymbol{v}_1$: the corresponding singular vector.

/* $\mathcal{N}(\mathbf{0}, \boldsymbol{I}_\ell)$ is the standard multivariate normal distribution */

$\boldsymbol{x}^{(0)} \leftarrow \mathcal{N}(\mathbf{0}, \boldsymbol{I}_\ell)$, $\boldsymbol{x}^{(0)} \leftarrow \frac{\boldsymbol{x}^{(0)}}{\left\|\boldsymbol{x}^{(0)}\right\|_2}$

for $i = 1$ **to** k **do**

  $\boldsymbol{x}^{(i)} \leftarrow \boldsymbol{A}^\top \boldsymbol{A} \boldsymbol{x}^{(i-1)}$, $\boldsymbol{x}^{(i)} \leftarrow \frac{\boldsymbol{x}^{(i)}}{\left\|\boldsymbol{x}^{(i)}\right\|_2}$

${\hat{\sigma}_1}^2 \leftarrow \left(\boldsymbol{x}^{(k)}\right)^\top \boldsymbol{A}^\top \boldsymbol{A} \boldsymbol{x}^{(k)}, \boldsymbol{v}_1 \leftarrow \boldsymbol{x}^{(k)}$

return ${\hat{\sigma}_1}^2, \boldsymbol{v}_1$

singular value σ_1^2 satisfies $|\hat{\sigma}_1^2 - \sigma_1^2| = O\left(\sigma_2^2/\sigma_1^2\right)^{2k}$ [31], where σ_2^2/σ_1^2 (*spectral gap*) is the ratio of the second-largest squared singular value of $\boldsymbol{A}$ to the top one.

However, instead of the *spectral-gap-dependent* error bound, the *spectral-gap-free* error bound (which does not depend on σ_2^2) of Power Iteration is more convenient for proving the theorems related to the AeroSketch framework proposed in this paper. A *spectral-gap-free* error bound was proved by Kuczyński and Woźniakowski as Theorem 4.1(a) in [17], which provides a tight bound but in a rather lengthy form. Therefore, we derive a looser yet more concise version that suffices for our analysis and state it as follows:

Corollary 2.1 (Probabilistic error bound and time complexity of Power Iteration, a simplified corollary of Theorem 4.1(a) in [17]). *For Algorithm 1, if we set $k = \lceil \log_2 d \rceil + 1$, then the probability that the estimated top squared singular value $\hat{\sigma}_1^2$ exceeds half of the true value $\sigma_1^2/2$ is bounded by:*

$$\Pr\left[\hat{\sigma}_1^2 \geq \sigma_1^2/2\right] \geq 1 - \frac{2}{\pi\sqrt{e}} \cdot \frac{1}{\sqrt{d \log_2 d}}.$$

Power Iteration requires only matrix-vector multiplications. The total computational time cost of Algorithm 1 is $O(d\ell k)$. If we set $k = \lceil \log_2 d \rceil + 1$, then the time cost becomes $O(d\ell \log d)$.

The original statement of Theorem 4.1(a) in [17], together with the detailed derivation of Corollary 2.1, can be found in our technical report [1].

2.2.2 Simultaneous Iteration. **Simultaneous Iteration**, also known as **Subspace Iteration** or **Orthogonal Iteration**, is an efficient algorithm for approximating the top-k singular values and vectors of a matrix (in contrast to Power Iteration, which approximates only the top-1) [27]. The procedure for this method is shown in Algorithm 2.

After running Algorithm 2 on matrix $\boldsymbol{A}$, we obtain the approximate top-k singular vectors $\boldsymbol{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_k]$ and the approximate top-k singular values $\hat{\Sigma} = \mathrm{diag}(\hat{\sigma}_1, \hat{\sigma}_2, \ldots, \hat{\sigma}_k)$ of $\boldsymbol{A}$. In more intuitive terms, if the exact singular value decomposition (SVD) of $\boldsymbol{A}$ is given by $\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$, then Algorithm 2 returns $\boldsymbol{Z} \approx \boldsymbol{U}_k$ and $\hat{\Sigma} \approx \Sigma_k$, where Σ_k is the diagonal matrix containing the top-k singular values of $\boldsymbol{A}$, and $\boldsymbol{U}_k$ contains the corresponding singular vectors.

The probabilistic error guarantees for the approximations, produced by Algorithm 2 provided by Musco and Musco [27], are sufficient for our later analysis and are stated as follows:

Theorem 2.2 (Probabilistic error bounds and time complexity of Simultaneous Iteration, the main lemma proven in [27]). *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{d\times\ell}$, a target rank k, and an error parameter*

Algorithm 2: Simultaneous Iteration (`simul_iter`)

Input: $\boldsymbol{A} \in \mathbb{R}^{d\times\ell}$: the matrix for which the top-k singular values and their corresponding singular vectors are to be obtained; k: the number of singular components to compute; ε_{SI}: an error coefficient within the range $(0, 1)$.

Output: $Z \in \mathbb{R}^{d\times k}$: the matrix of the approximate top-k singular vectors; $\hat{\Sigma} \in \mathbb{R}^{k\times k}$: the diagonal matrix of the approximate top-k singular values.

1 $q \leftarrow \Theta\left(\frac{\log d}{\varepsilon_{\text{SI}}}\right), \boldsymbol{\Pi} \sim \mathcal{N}(0,1)^{\ell\times k}$

2 $\boldsymbol{K} \leftarrow (\boldsymbol{A}\boldsymbol{A}^\top)^q \boldsymbol{A}\boldsymbol{\Pi}$ // $\boldsymbol{K} \in \mathbb{R}^{d\times k}$, takes $\Theta(qd\ell k)$ time

3 $\boldsymbol{Q}, \boldsymbol{R} \leftarrow \mathtt{QR}(\boldsymbol{K})$ // $\boldsymbol{Q} \in \mathbb{R}^{d\times k}$, takes $\Theta\left(dk^2\right)$ time

4 $\boldsymbol{M} \leftarrow \boldsymbol{Q}^\top \boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{Q}$ // $\boldsymbol{M} \in \mathbb{R}^{k\times k}$, takes $O(d\ell k)$ time

5 $[\boldsymbol{U}, \hat{\Sigma}^2, \boldsymbol{U}^\top] \leftarrow \mathtt{SVD}(\boldsymbol{M})$ // takes $O(k^3)$ time

6 **return** $Z \leftarrow \boldsymbol{Q}\boldsymbol{U}$, and $\hat{\Sigma}$

ε_{SI}, after executing Algorithm 2, we obtain matrices $\boldsymbol{Z} \in \mathbb{R}^{d\times k}$ and $\hat{\boldsymbol{\Sigma}} \in \mathbb{R}^{k\times k}$. With high probability (at least 99/100), the following guarantees hold:

(1) $\|\boldsymbol{A} - \boldsymbol{Z}\boldsymbol{Z}^\top\boldsymbol{A}\|_F \le (1+\varepsilon_{SI})\|\boldsymbol{A} - \boldsymbol{A}_k\|_F$,
where $\boldsymbol{A}_k$ is the best rank-k approximation of $\boldsymbol{A}$. If the singular value decomposition of $\boldsymbol{A}$ is $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, then $\boldsymbol{A}_k = \boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^\top$. [1]

(2) $\|\boldsymbol{A} - \boldsymbol{Z}\boldsymbol{Z}^\top\boldsymbol{A}\|_2 \le (1+\varepsilon_{SI})\|\boldsymbol{A} - \boldsymbol{A}_k\|_2$.

(3) $\forall i$, $\left|\boldsymbol{u}_i^\top\boldsymbol{A}\boldsymbol{A}^\top\boldsymbol{u}_i - \boldsymbol{z}_i^\top\boldsymbol{A}\boldsymbol{A}^\top\boldsymbol{z}_i\right| \le \varepsilon_{SI}\cdot\sigma_{k+1}^2$, where $\boldsymbol{u}_i$ is the i-th left singular vector of $\boldsymbol{A}$ and $\boldsymbol{z}_i$ is the i-th column of $\boldsymbol{Z}$.

The comments in Algorithm 2 analyze the time cost of each line. The time complexity of Algorithm 2 is dominated by line 2, which takes $\Theta(qd\ell k)$ time, where $q = \Theta\left(\frac{\log d}{\varepsilon_{SI}}\right)$. If ε_{SI} is a constant, then the total time complexity of Algorithm 2 becomes $\Theta(d\ell k \log d)$.

2.3 Frequent Directions

Frequent Directions (FD) [8, 20] is a deterministic matrix sketch algorithm in the row-update model. It solves Problem 1 with optimal space and update time complexity. At initialization, FD creates an all-zero matrix $\boldsymbol{B}$ with the size of $2\ell \times d$, where ℓ is an input parameter satisfying $\ell \le d/2$. To process each arriving row vector $\boldsymbol{a}_i$, FD first checks whether $\boldsymbol{B}$ contains any zero-valued rows. If so, it inserts $\boldsymbol{a}_i$ into one of them. Otherwise, all 2ℓ rows are filled, and it performs a singular value decomposition $[\boldsymbol{U}, \Sigma, \boldsymbol{V}^\top] = \mathtt{SVD}(\boldsymbol{B})$, rescales the "directions" in $\boldsymbol{B}$ using the ℓ-th largest singular value σ_ℓ, and "forgets" the least significant direction in the column space of $\boldsymbol{V}^\top$. The updated Σ' and sketch matrix $\boldsymbol{B}'$ are computed as:

$$\Sigma' = \operatorname{diag}\left(\sqrt{\sigma_1^2 - \sigma_\ell^2}, \ldots, \sqrt{\sigma_{\ell-1}^2 - \sigma_\ell^2}, 0, \ldots, 0\right),$$

and $\boldsymbol{B}' = \Sigma'\boldsymbol{V}^\top$, where $\sigma_1^2 \ge \sigma_2^2 \ge \cdots \ge \sigma_{2\ell}^2$. The updated $\boldsymbol{B}'$ will have at least $\ell + 1$ zero rows, allowing the process to continue updating the sketch matrix $\boldsymbol{B}$ as new row vectors arrive. The amortized computational cost per row is $O(d\ell)$.

Frequent Directions has also been adapted to the matrix sketching variants defined in Section 2.1—for example, PFD [30] for Problem 3 (ATTP Sketch) and DS-FD [44] for Problem 2 (Matrix Sketching over Sliding Window). However, a common drawback of these methods is that they

[1] Specifically, for $k = 0$, we define $\boldsymbol{A}_0 = \boldsymbol{0}$.

fail to achieve the optimal update time complexity of the original Frequent Directions algorithm, especially under tight approximation error constraints ε. In this paper, we develop a new framework with theoretical guarantees, termed AeroSketch, to overcome this limitation for Problems 2, 3, 4, 5, and 6.

3 Our Framework: AeroSketch

In this section, we present a framework, termed AeroSketch, for accelerating the update time complexity of the baseline algorithms solving the tracking matrix sketching problems introduced in Section 2.1. We first select Problem 2 as a representative case and describe how our AeroSketch framework is applied. We also provide rigorous end-to-end theoretical proofs for the error bound, time, and space complexity. In later sections, we show how the AeroSketch framework can be applied to optimize other problems.

3.1 AeroSketch Framework Description

3.1.1 High-level idea. Taking Problem 2 as an example, AeroSketch shares a common high-level idea and skeleton with Fast-DS-FD: we implicitly maintain a matrix sketch $\boldsymbol{B}_{\mathrm{W}}$ to approximate the matrix $\boldsymbol{A}_{\mathrm{W}}$ consisting of the most recent N elements in the stream. As time progresses, matrix $\boldsymbol{A}_{\mathrm{W}}$ continuously changes as new vectors arrive and the old vectors slide out of the window. When the difference between the matrix $\boldsymbol{A}_{\mathrm{W}}$ and the sketch matrix $\boldsymbol{B}_{\mathrm{W}}$ exceeds the error threshold θ, this discrepancy is computed, and a **snapshot** containing information about the difference and the current timestamp is recorded. As the sliding window advances, snapshots with old timestamps expire. The remaining recorded snapshots can be summed to form the sketch matrix $\boldsymbol{B}_{\mathrm{W}}$ that approximates matrix $\boldsymbol{A}_{\mathrm{W}}$ within the current sliding window.

In Fast-DS-FD [44], the difference between $\boldsymbol{A}_{\mathrm{W}}$ and $\boldsymbol{B}_{\mathrm{W}}$ is computed by singular value decomposition (SVD). It calculates the singular values and vectors of the covariance residual matrix $\boldsymbol{C}^\top\boldsymbol{C} = \boldsymbol{A}_{\mathrm{W}}^\top\boldsymbol{A}_{\mathrm{W}} - \boldsymbol{B}_{\mathrm{W}}^\top\boldsymbol{B}_{\mathrm{W}}$, and components where the singular values are greater than θ are recorded as snapshots. If we set $\theta = O(1)\cdot\varepsilon\left\|\boldsymbol{A}_{\mathrm{W}}\right\|_F^2$, then we will have

$$\left\|\boldsymbol{A}_{\mathrm{W}}^\top\boldsymbol{A}_{\mathrm{W}} - \boldsymbol{B}_{\mathrm{W}}^\top\boldsymbol{B}_{\mathrm{W}}\right\|_2 = \left\|\boldsymbol{C}^\top\boldsymbol{C}\right\|_2 \le \theta = O(1)\cdot\varepsilon\left\|\boldsymbol{A}_{\mathrm{W}}\right\|_F^2,$$

which satisfies the covariance error bound. Since $\boldsymbol{A}_{\mathrm{W}}$ (consequently $\boldsymbol{C}$) changes with each incoming row vector, there may be new singular values greater than θ. Therefore, SVD must be performed on $\boldsymbol{C}^\top\boldsymbol{C}$ for each update. When $1/\varepsilon > d$, the residual matrix $\boldsymbol{C}$ reaches a size of $d\times d$, and each SVD update will cost $O(d^3)$ time. For high-dimensional vector streams, this complexity is unacceptable.

As mentioned in the introduction, AeroSketch uses RandNLA operators, such as Power Iteration and Simultaneous Iteration, to overcome the time bottleneck caused by SVD. This is because we only need singular values greater than the error threshold θ, while the time-consuming full SVD computes all singular values, resulting in computational waste. However, since RandNLA methods are approximate algorithms, directly summing the approximate snapshots using the method of Fast-DS-FD would introduce significant **cumulative restoring norm error** in the $\boldsymbol{B}_{\mathrm{W}}$, in the form of

$$\left\|\sum_{i=T-N+1}^{T}\left(\boldsymbol{Z}_i\boldsymbol{Z}_i^\top{\boldsymbol{C}'_i}^\top\boldsymbol{C}'_i + \boldsymbol{C}'_i{}^\top\boldsymbol{C}'_i\boldsymbol{Z}_i\boldsymbol{Z}_i^\top - 2\boldsymbol{Z}_i\boldsymbol{Z}_i^\top{\boldsymbol{C}'_i}^\top\boldsymbol{C}'_i\boldsymbol{Z}_i\boldsymbol{Z}_i^\top\right)\right\|_2. \tag{1}$$

If $\boldsymbol{Z}_i$'s are formed by exact singular vectors of ${\boldsymbol{C}'_i}^\top\boldsymbol{C}'_i$ as in Fast-DS-FD, then Eq. (1) is implicitly zero. However, if the $\boldsymbol{Z}_i$'s are approximate singular vectors, the restoring norm error becomes explicitly nonzero and can accumulate as the update proceeds, as shown by the blue line in Figure 1 (See our technical report [1] for a detailed derivation). It is also difficult to derive an upper bound for this error. Fortunately, we found that by redesigning the saved snapshot ($\boldsymbol{Z}_i$ and $\boldsymbol{Z}_i^\top{\boldsymbol{C}'_i}^\top\boldsymbol{C}'_i$ at

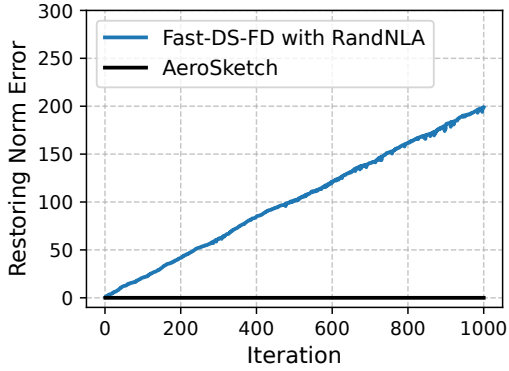


Fig. 1. Directly replacing the SVD in Fast-DS-FD with RandNLA would introduce cumulative restoring norm errors as shown in Eq. (1), whereas our AeroSketch does not.

line 15 of Algorithm 4) and the restoring method of $\boldsymbol{B}_{\mathrm{W}}$ involving the snapshots (Algorithm 5), this cumulative error can be eliminated. Furthermore, other execution procedures and parameter selections must be meticulously configured to bound the error, space/communication costs, and time complexity. Through analysis and experiments, we find that the restoring norm error becomes zero (as shown by the black line in Figure 1) and the algorithm's error now stems solely from the FD reduce operation and the residual matrix $\boldsymbol{C}_{T-N}$ at time $T-N$, both of which are more tractable for error analysis. Additionally, we discovered that the auxiliary sketch and queue in Fast-DS-FD can be removed, saving their space and time overhead.

Algorithm 3: AeroSketch: Initialize(d, ε, θ)

Input: d: dimension of input vectors of Update; ε: error upper bound; θ: dump threshold.

1 $\ell \leftarrow \lceil \frac{2}{\varepsilon} \rceil, \quad \boldsymbol{C}_0 \leftarrow \boldsymbol{0}_{2\ell \times d}$

2 $\mathcal{S} \leftarrow$ queue.Initialize() `// Queue of snapshots`

3.1.2 Data Structures. Algorithm 3 shows the data structures used by AeroSketch. At initialization, AeroSketch requires the input vector dimension d, the upper bound on the relative covariance error ε, and a dump threshold θ. It then initializes an empty matrix $\boldsymbol{C}_0$ of size $2\ell \times d$ and a queue of snapshots $\mathcal{S}$.

3.1.3 Update Algorithm. Algorithm 4 shows how AeroSketch updates the sketch matrix $\boldsymbol{C}_i$ when a new row vector $\boldsymbol{a}_i$ arrives at timestamp i. First, we insert $\boldsymbol{a}_i$ into an empty row of $\boldsymbol{C}_{i-1}$ to form $\tilde{\boldsymbol{C}}_i$ (line 1). If the number of rows in $\tilde{\boldsymbol{C}}_i$ reaches 2ℓ, we apply lines 3-4, the FD reduction described in Section 2.3 to form $\boldsymbol{C}'_i$. Otherwise, we simply set $\boldsymbol{C}'_i$ to $\tilde{\boldsymbol{C}}_i$ (line 6). Next, we perform Power Iteration on matrix $\boldsymbol{C}'_i$ to estimate the squared top singular value $\hat{\sigma}_1^2$ (line 7). If it exceeds half the dump threshold, i.e., $\hat{\sigma}_1^2 > \theta/2$ (line 8), we apply Simultaneous Iteration to $\boldsymbol{C}'_i$ to compute the approximate squared top-k singular values greater than θ along with the corresponding singular vectors. Thus, instead of recomputing the exact singular vectors, we quickly approximate them in a few iterations.

However, there is a catch: before applying Simultaneous Iteration, we do not know how many squared singular values are greater than θ, and thus we cannot predefine k, which is a required parameter of Simultaneous Iteration. To resolve this, we use the classic *doubling* technique. We

Algorithm 4: AeroSketch: Update($\boldsymbol{a}_i$)

Input: $\boldsymbol{a}_i$: the row vector arriving at timestamp i.

1 Insert $\boldsymbol{a}_i$ into an empty row of $\boldsymbol{C}_{i-1}$ to get $\tilde{\boldsymbol{C}}_i$
2 **if** $rows(\tilde{\boldsymbol{C}}_i) == 2\ell$ **then** // $\tilde{\boldsymbol{C}}_i$ has no zero rows
3 $[\boldsymbol{U}_i, \Sigma_i, \boldsymbol{V}_i^\top] \leftarrow \mathtt{SVD}(\tilde{\boldsymbol{C}}_i)$ // $\Sigma_i = \mathtt{diag}\left(\sigma_1, \sigma_2 \ldots, \sigma_{\min(2\ell,d)}\right)$
4 $\boldsymbol{C}'_i \leftarrow \sqrt{\max\left(\Sigma_i^2 - \sigma_\ell^2 \boldsymbol{I}, \boldsymbol{0}\right)}\, \boldsymbol{V}_i^\top$ // element-wise max
5 **else**
6 $\boldsymbol{C}'_i \leftarrow \tilde{\boldsymbol{C}}_i$
7 $\hat{\sigma}_1^2, \hat{\boldsymbol{v}}_1 \leftarrow \mathtt{power\_iteration}(\boldsymbol{C}_i'^\top, k = \lceil \log_2 d \rceil + 1)$
8 **if** $\hat{\sigma}_1^2 \geq \frac{\theta}{2}$ **then**
9 **for** $j = 1$ **to** $\lceil \log_2 \min(2\ell, d) \rceil$ **do**
10 $k \leftarrow \min(2^j, 2\ell, d)$
11 $[\boldsymbol{Z}'_i, \hat{\Sigma}] \leftarrow \mathtt{simul\_iter}(\boldsymbol{C}_i'^\top, k, \varepsilon_{\mathrm{SI}} = 0.4)$ // $\hat{\Sigma} = \mathtt{diag}\left(\hat{\sigma}_1, \hat{\sigma}_2 \ldots, \hat{\sigma}_k\right)$
12 **if** $\hat{\boldsymbol{\sigma}}_k^2 < \theta$ **then**
13 $\xi \leftarrow \max_\xi\{\hat{\boldsymbol{\sigma}}_\xi^2 \geq \theta\}$
14 $\boldsymbol{Z}_i \leftarrow \boldsymbol{Z}'_i[:, 1:\xi\,]$
15 $\mathcal{S}$.append$((\boldsymbol{Z}_i, \boldsymbol{Z}_i^\top \boldsymbol{C}_i'^\top \boldsymbol{C}'_i), s = \mathcal{S}[-1].t + 1, t = i)$ // $\mathcal{S}[-1]$ is the tail element in queue $\mathcal{S}$
16 $\boldsymbol{C}_i \leftarrow \boldsymbol{C}'_i - \boldsymbol{C}'_i \boldsymbol{Z}_i \boldsymbol{Z}_i^\top$
17 break
18 $\boldsymbol{C}_i \leftarrow \boldsymbol{C}'_i$
19 **else**
20 $\boldsymbol{C}_i \leftarrow \boldsymbol{C}'_i$

start by computing the top-2 approximate singular values and vectors of $\boldsymbol{C}'_i$ by setting $k = 2$ in Simultaneous Iteration, and then continue doubling k (i.e., $k = 4, 8, 16, \ldots, 2^j$, as in lines 9-11). Eventually, we find the value $k = 2^j$ such that the k-th top estimated squared singular value is smaller than θ (line 12). This implies the existence of some ξ satisfying $2^{j-1} \leq \xi < 2^j$ such that the ξ-th largest estimated squared singular value is at least θ, while the $(\xi+1)$-th and smaller estimated squared singular values fall below θ (line 13).

Once ξ is determined, we extract the approximate top-ξ singular vectors as a matrix denoted by $\boldsymbol{Z}_i$ (line 14). We then save a snapshot containing two matrices $\boldsymbol{Z}_i$ and $\boldsymbol{Z}_i^\top \boldsymbol{C}_i'^\top \boldsymbol{C}'_i$ along with the current timestamp i, and push it into the queue $\mathcal{S}$ (line 15). Next, we remove the subspace spanned by $\boldsymbol{Z}_i$ from $\boldsymbol{C}'_i$ to obtain $\boldsymbol{C}_i$ (line 16). Finally, the for-loop and the update procedure terminate (line 17).

3.1.4 Query Algorithm. Algorithm 5 shows how to construct the sketch matrix $\hat{\boldsymbol{B}}$ for the current sliding window $(T-N, T]$. Since we store snapshots of the form $(\boldsymbol{Z}_i, \boldsymbol{Z}_i^\top \boldsymbol{C}_i'^\top \boldsymbol{C}'_i)$ for certain timestamps i in the snapshot queue $\mathcal{S}$, we can compute $\boldsymbol{Z}_i \boldsymbol{Z}_i^\top \boldsymbol{C}_i'^\top \boldsymbol{C}'_i + \boldsymbol{C}_i'^\top \boldsymbol{C}'_i \boldsymbol{Z}_i \boldsymbol{Z}_i^\top - \boldsymbol{Z}_i \boldsymbol{Z}_i^\top \boldsymbol{C}_i'^\top \boldsymbol{C}'_i \boldsymbol{Z}_i \boldsymbol{Z}_i^\top$ by matrix multiplications and additions. We then sum it over all i within the sliding window and add them to the residual matrix $\boldsymbol{C}_T^\top \boldsymbol{C}_T$ to obtain matrix $\boldsymbol{B}$, as shown in lines 1-4. Finally, we perform an eigendecomposition of $\boldsymbol{B}$ (line 5), reduce it by the ℓ-th largest eigenvalue (line 6), and return the top-ℓ rows of the resulting matrix as the sketch matrix $\hat{\boldsymbol{B}} \in \mathbb{R}^{\ell \times d}$ (line 7).

Algorithm 5: AeroSketch: Query(lb, ub)

Input: lb, ub: start and end timestamp of the query. For Problem 2, $lb = T - N$, $ub = T$.

$\boldsymbol{B} \leftarrow \boldsymbol{C}_{ub}^\top \boldsymbol{C}_{ub}$
forall $((Z_j, Z_j^\top C_j'^\top C_j'), s, t = j)$ *of* $\mathcal{S}$ **do**
  if $lb < t = j \leq ub$ **then**
    $\boldsymbol{B} \leftarrow \boldsymbol{B} + Z_j Z_j^\top C_j'^\top C_j' + C_j'^\top C_j' Z_j Z_j^\top - Z_j Z_j^\top C_j'^\top C_j' Z_j Z_j^\top$
$[\boldsymbol{V}, \boldsymbol{\Lambda}, \boldsymbol{V}^\top] \leftarrow$ `eigen_decomposition(`$\boldsymbol{B}$`)` // $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2 \ldots, \lambda_d), \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$
$\hat{\boldsymbol{B}} \leftarrow \sqrt{\max(\boldsymbol{\Lambda} - \boldsymbol{I}\lambda_\ell, \boldsymbol{0})}\, \boldsymbol{V}^\top$
return $\hat{\boldsymbol{B}}[1:\ell, :]$

Algorithm 6: ML-AeroSketch

Function Initialize(d, N, R, ε)**:**
  Input: d: dimension of the vectors in the stream; N: sliding window size; R: upper bound of squared norms; ε: desired relative covariance error bound.
  $L \leftarrow \lceil \log_2 R \rceil$; $M \leftarrow []$
  for $i = 0$ **to** $L - 1$ **do**
    M.append(AeroSketch.Initialize($d, \varepsilon, \theta = 2^i \varepsilon N$))
Function Update($\boldsymbol{a}_i$)**:**
  Input: $\boldsymbol{a}_i$: the row vector arriving at timestamp i.
  for $j = 0$ **to** $L - 1$ **do**
    while $len(M[j].\mathcal{S}) > \frac{8}{\varepsilon}$ *or* $M[j].\mathcal{S}[0].t \leq i - N$ **do**
      $M[j].\mathcal{S}$.Popleft()
    if $\|\boldsymbol{a}_i\|_2^2 \geq \theta$ **then**
      $M[j].\mathcal{S}$.append($\boldsymbol{a}_i, s = M[j].\mathcal{S}[-1].t, t = i$)
    else
      $M[j]$.Update($\boldsymbol{a}_i$)
Function Query()**:**
  $j \leftarrow \min_j \{1 \leq M[j].\mathcal{S}[0].s \leq T - N + 1\}$
  return FD($M[j]$.Query($lb = T - N, ub = T$), $[\boldsymbol{a}_i \in M[j].\mathcal{S}]$)

3.1.5 General Unnormalized Model. For the actual implementation of AeroSketch for Problem 2, we adopt the *parallel multi-level* technique, as described in Algorithm 6. Briefly, we use this technique for two main reasons: (1) As new rows arrive and the sliding window advances, old row vectors (those arriving before time $T-N$) become outdated. Therefore, we first check for and remove expired snapshots from the queue $\mathcal{S}$ (line 7). (2) We have to set the parameter $\theta = O(1) \cdot \varepsilon \left\|\boldsymbol{A}_{T-N,T}\right\|_F^2$ at initialization, but $\left\|\boldsymbol{A}_{T-N,T}\right\|_F^2$ can vary dramatically between very small and very large values as the sliding window progresses. To handle this variability, we instantiate multiple parallel AeroSketch levels (lines 3-4). This multi-level sketching strategy is widely used in streaming algorithms under the sliding window model [19, 40, 42, 44]. This technique guarantees that there will always be at least one level j for which the error threshold satisfies $\theta \leq \varepsilon \left\|\boldsymbol{A}_{T-N,T}\right\|_F^2$. Specifically:

Fact 1. *There always exists one level* $j = \lfloor \log_2 \frac{\|\boldsymbol{A}_{T-N,T}\|_F^2}{N} \rfloor$ *such that*

$$\theta = 2^j \varepsilon N \leq \varepsilon \|\boldsymbol{A}_{T-N,T}\|_F^2 \leq 2\theta,$$

and we can find the required j *using line 14 of Algorithm 6, in the same way as Algorithm 7 does in [44].*

3.2 Analysis

We provide the error bound, asymptotic space and time complexities of Algorithm 6 in Theorem 3.1.

Theorem 3.1. *ML-AeroSketch (Algorithm 6) solves the Tracking Approximate Matrix Sketch over Sliding Window (Problem 2) with space complexity* $O\left(\frac{d}{\varepsilon}\log R\right)$ *and amortized time complexity* $O\left(\frac{d}{\varepsilon}\log d \log R\right)$ *per update, with success probability (the probability that the output sketch satisfies the covariance error bound) at least*

$$\frac{99}{100}\left(1 - \frac{2}{\pi\sqrt{e}} \cdot \frac{1}{\sqrt{d\log_2 d}}\right). \tag{2}$$

When d is not very small, the success probability lower bound Eq. (2) is close to $\frac{99}{100}$, which is a high probability.[2] We plot the function graph of the success probability lower bound Eq. (2) with respect to the vector dimension d in Figure 2. As the dimension d increases, the success probability lower bound of the algorithm quickly approaches $\frac{99}{100}$. When $d \geq 6$, the success probability is at least 0.9, and when $d \geq 20$ and $d \geq 192$, the success probability is at least 0.95 and 0.98, respectively. This indicates that when d is not very small, the success probability lower bound is close to $\frac{99}{100}$.

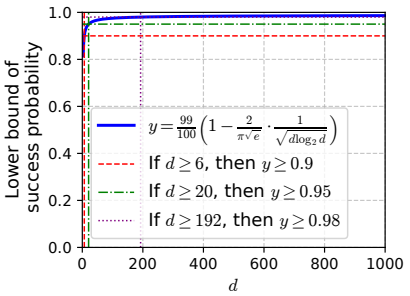


Fig. 2. Success probability lower bound with respect to d.

3.2.1 Reducing the Failure Probability from Eq. (2) to Arbitrarily Small δ. Furthermore, with a simple modification to Algorithm 4, we can boost the success probability of the algorithm from at most 99/100 in Eq. (2) to $1 - \delta$, where δ can be made arbitrarily close to 0. At the same time, the time complexity only increases by a multiplicative factor of $\log^2(1/\delta)$ compared with the result in Theorem 3.1, i.e., it increases from $O\left(\frac{d}{\varepsilon}\log d \log R\right)$ to $O\left(\frac{d}{\varepsilon}\log d \log R \log^2(1/\delta)\right)$, while the space complexity remains the same as in Theorem 3.1.

More specifically, this probability-amplification technique replaces line 11 of Algorithm 4 with Algorithm 7. The key idea is to replace the original single Simultaneous Iteration with $r = \log(1/\delta)$ independent Simultaneous Iterations, and then select the iteration whose result minimizes the matrix spectral norm of the $\boldsymbol{C}_i' - \boldsymbol{C}_i'\boldsymbol{Z}\boldsymbol{Z}^\top$. Formally, select Z_{r^*} where

$$r^* = \operatorname*{argmin}_{h\in[r]} \left\|\boldsymbol{C}_i' - \boldsymbol{C}_i'\boldsymbol{Z}_h\boldsymbol{Z}_h^\top\right\|_2.$$

However, there is a pitfall here: computing the matrix spectral norm $\left\|\boldsymbol{C}_i' - \boldsymbol{C}_i'\boldsymbol{Z}_h\boldsymbol{Z}_h^\top\right\|_2$ exactly in each round incurs significant time overhead. Therefore, instead of computing the exact spectral

[2]Note that this is a "lower bound" of the success probability, and the actual success probability could be much higher. In fact, in our real experiments, there were no instances where the output sketch failed to satisfy the covariance error bound.

norm, we repeatedly run $s = 2\log_3 \frac{2}{\delta}$ independent rounds of Power Iteration on matrix $C_i' - C_i' Z_h Z_h^\top$ to obtain s estimates (lines 6-7), take the maximum among them as the estimate of $\left\|C_i' - C_i' Z_h Z_h^\top\right\|_2$ (line 8), and finally choose the round with the smallest estimated value among all r rounds as the final result (line 9). It can be shown that, after this modification, we obtain the following corollary:

Corollary 3.2. *If Algorithm 7 is applied, then ML-AeroSketch (Algorithm 6) solves the Tracking Approximate Matrix Sketch over Sliding Window (Problem 2) with space complexity* $O\left(\frac{d}{\varepsilon}\log R\right)$ *and amortized time complexity* $O\left(\frac{d}{\varepsilon}\log d \log R \log^2 \frac{1}{\delta}\right)$ *per update, with success probability at least* $1-\delta$*, where* δ *is a tunable parameter that can be chosen arbitrarily in the interval* $(0, 1/100)$.

The detailed proof can be found in our technical report [1].

Algorithm 7: Probability Amplification Procedure (Replacing Line 11 in Algorithm 4)

1 $c \leftarrow +\infty, \quad r \leftarrow \log_{100} \frac{2}{\delta}, \quad s \leftarrow 2\log_3 \frac{2}{\delta}$
2 **for** $h = 1$ **to** r **do**
3 $\quad [Z_h', \hat{\Sigma}_h] \leftarrow \texttt{simul\_iter}(C_i'^\top, k, \varepsilon_{\text{SI}} = 0.2)$ // $\hat{\Sigma}_h = \texttt{diag}\,(\hat{\sigma}_{h1}, \hat{\sigma}_{h2} \ldots, \hat{\sigma}_{hk})$
4 $\quad \xi_h \leftarrow \max_\xi\{\hat{\sigma}_{h\xi}^2 \geq \theta\}, \quad Z_h \leftarrow Z_h'[:, 1:\xi]$
5 $\quad c_h \leftarrow 0$
6 $\quad$ **for** $g = 1$ **to** s **do**
7 $\quad\quad c_{hg}^2 \leftarrow \texttt{power\_iteration}(C_i' - C_i' Z_h Z_h^\top)$
8 $\quad\quad c_h \leftarrow \max(c_h, c_{hg})$
9 $\quad$ **if** $c_h < c$ **then** $c \leftarrow c_h, \quad Z_i' \leftarrow Z_h'$

3.2.2 A proof sketch of Theorem 3.1. For the complete proof of Theorem 3.1, we refer the reader to our technical report [1]. Here, we provide a concise proof sketch that contains the key technical ideas and intuition.

(Error bound) If $\|\boldsymbol{a}_i\|_2^2 \geq \theta$, then at line 10 in Algorithm 6, $\boldsymbol{a}_i$ is saved accurately, and thus no error is introduced. Otherwise, if line 12 is triggered, then Algorithm 4 is executed. Since line 15 of Algorithm 4 stores matrix Z_i and $Z_i^\top C_i'^\top C_i'$, we can accurately restore the matrix C_i' at line 4/6 from the matrix C_i at line 16 through the following computation:

$$\begin{aligned}
&C_i^\top C_i + \underline{Z_i} \cdot \underline{Z_i^\top C_i'^\top C_i'} + (\underline{Z_i^\top C_i'^\top C_i'})^\top \underline{Z_i}^\top - \underline{Z_i} \cdot \underline{Z_i^\top C_i'^\top C_i'} \cdot \underline{Z_i} \cdot \underline{Z_i}^\top \\
&= (C_i' - C_i' Z_i Z_i^\top)^\top (C_i' - C_i' Z_i Z_i^\top) + Z_i Z_i^\top C_i'^\top C_i' + C_i'^\top C_i' Z_i Z_i^\top - Z_i Z_i^\top C_i'^\top C_i' Z_i Z_i^\top \\
&= C_i'^\top C_i' \cancel{- Z_i Z_i^\top C_i'^\top C_i' - C_i'^\top C_i' Z_i Z_i^\top + Z_i Z_i^\top C_i'^\top C_i' Z_i Z_i^\top} \\
&\quad \cancel{+ Z_i Z_i^\top C_i'^\top C_i' + C_i'^\top C_i' Z_i Z_i^\top - Z_i Z_i^\top C_i'^\top C_i' Z_i Z_i^\top} = C_i'^\top C_i'.
\end{aligned}$$

This can be regarded as the "inverse operation" of Algorithm 4 and used in the query procedure of Algorithm 5. Therefore, lines 7-20 do not introduce the restoring norm error. The overall error mainly comes from the FD reduction operation in line 4. Additionally, by performing N steps of the "inverse operation" from the current time T, we can recover the matrix C_{T-N}' and thus C_{T-N} at time $T-N$. We can prove that the covariance error can be decomposed into two parts:

$$\left\|A_{\text{W}}^\top A_{\text{W}} - B^\top B\right\|_2 \leq \underbrace{\left\|\sum_{i=T-N+1}^{T} \Delta_i\right\|_2}_{\text{Term 1}} + \underbrace{\left\|C_{T-N}^\top C_{T-N}\right\|_2}_{\text{Term 2}}.$$

Among them, $\Delta_i = \tilde{C}_i^\top \tilde{C}_i - C_i'^\top C_i'$ is the gap error before and after line 4 in Algorithm 4. Therefore, Term 1 is the error that originates from the FD reductions, and Term 2 is the error from C_{T-N}. The derivation of the upper bound for Term 1 is similar to the idea of FD [8], and we can conclude that:

$$\text{Term 1} \le O(1)\varepsilon \left\|A_{T-N,T}\right\|_F^2 + 2 \cdot \text{Term 2}.$$

For the upper bound of Term 2 to hold, both the Power Iteration (line 7) and the Simultaneous Iteration (line 11) at time $T-N$ need to have converged successfully. According to Corollary 2.1 and Theorem 2.2, the probability is given by Eq. (2). Under the condition of successful convergence and based on Fact 1, we can always find one level j of Algorithm 6 that satisfies:

$$\text{Term 2} \le O(1)\theta \le O(1)\varepsilon \|A_{T-N,T}\|_F^2.$$

Finally, we have the probabilistic upper bound of covariance error:

$$\left\|A_{T-N,T}^\top A_{T-N,T} - B^\top B\right\|_2 \le \text{Term 1} + \text{Term 2} \le O(1)\varepsilon \|A_{T-N,T}\|_F^2.$$

(Time complexity) In Algorithm 6, the update function executes the update subroutines of Algorithm 4 at most $L = \lceil \log_2 R \rceil$ times. The time cost of Algorithm 4 consists of 3 parts: the FD reduction in lines 2-6, the Power Iteration in line 7, and the doubling Simultaneous Iteration in lines 8-17. The SVD for FD reduction is executed once every ℓ updates, and the amortized time cost of lines 2-6 is $O(d\ell^2)/\ell = O(d\ell)$. According to the time complexity result in Corollary 2.1, the Power Iteration in line 7 takes $O(d\ell k) = O(\frac{d}{\varepsilon}\log d)$ time.

The analysis of the time cost for lines 9 to 17 is more subtle. We need to analyze each value of k when the for-loop terminates during the i-th update, which we denote as k_i. k_i is an integer, where intuitively each 1 in k_i corresponds to a component greater than θ in $\left\|A_{T-N,T}\right\|_F^2$. Therefore, for level j given by Fact 1, we have set $\theta = \varepsilon \left\|A_{T-N,T}\right\|_F^2$, and a constraint on the sum of k_i within the current sliding window holds:

$$\sum_{i=T-N+1}^{T} k_i = O\left(\frac{\left\|A_{T-N,T}\right\|_F^2}{\theta}\right) = O\left(\frac{1}{\varepsilon}\right). \tag{3}$$

According to Theorem 2.2, the time cost of the Simultaneous Iteration corresponding to each k_i is $O(d\ell k_i \log d)$. Therefore, the total time cost for all i is:

$$\sum_{i=T-N+1}^{T} O(d\ell k_i \log d) = O(d\ell \log d) \sum_{i=T-N+1}^{T} k_i = O\left(\frac{d\ell}{\varepsilon}\log d\right).$$

We assume $N > \ell$ (otherwise, if $N < \ell$, we can directly store all ℓ rows exactly, so the assumption $N \ge \ell$ is without loss of generality). Amortizing over N updates, the per-update time cost of Algorithm 4 of level j becomes:

$$\frac{1}{N} O\left(\frac{d\ell}{\varepsilon}\log d\right) = O\left(\frac{d}{\varepsilon}\log d\right).$$

We show that the update time complexity of level $j = \lfloor \log_2 \frac{\|A_{T-N,T}\|_F^2}{N} \rfloor$ in Algorithm 6 given by Fact 1 is the maximum over all L levels. The intuition is that for lower levels than j, the θ decreases exponentially, so the time-efficient line 10 can be triggered more frequently. For levels higher than j, the θ increases exponentially, so the sum of k_i in Eq. (3) decreases exponentially. Therefore, the update time cost is dominated by the level j, and the total time cost of all $L = \lceil \log_2 R \rceil$ levels is $O\left(\frac{d}{\varepsilon}\log d \log R\right)$.

(Space complexity) The space cost of each level in Algorithm 6 consists of the sketch matrix C and the snapshots in $\mathcal{S}$. The sketch matrix C requires $O(d\ell)$ space, as its maximum size is $d \times 2\ell$.

The queue $\mathcal{S}$ also uses $O(d/\varepsilon)$ space, since it stores matrices $\boldsymbol{Z}_i \in \mathbb{R}^{d\times k_i}$ and $\boldsymbol{Z}_i^\top \boldsymbol{C}_i'^\top \boldsymbol{C}_i' \in \mathbb{R}^{k_i\times d}$. Summing over all N updates and using the bound from Eq. (3), the space cost of the queue $\mathcal{S}$ is:

$$O\left(d \sum_{i=T-N+1}^{T} k_i\right) = O\left(\frac{d}{\varepsilon}\right).$$

Thus, the space cost of each level is $O\left(d\ell + \frac{d}{\varepsilon}\right) = O\left(\frac{d}{\varepsilon}\right)$, and the total space cost of all $L = \lceil \log_2 R \rceil$ levels is $O\left(\frac{d}{\varepsilon}\log R\right)$.

4 Applying AeroSketch to Other Problems

Now we show how AeroSketch serves as a general framework that can be plugged into various algorithms to replace their time-consuming matrix factorizations for a broad class of matrix sketching problems.

4.1 Persistent Matrix Sketch (Problem 3)

Algorithm 8 describes how AeroSketch can be used to solve the ATTP matrix sketch problem. The high-level idea is similar to FD-ATTP (Algorithm 1 in [30]), but we replace the SVD calculation with AeroSketch to avoid frequent and time-consuming matrix factorizations.

Algorithm 8: ATTP Sketch using AeroSketch Framework

```
Function Initialize(d, ε):
    ‖A_0‖_F^2 ← 0
    AeroSketch.Initialize(d, ε, θ = 0)
Function Update(a_i):
    ‖A_t‖_F^2 ← ‖A_{t−1}‖_F^2 + ‖a_i‖_2^2
    AeroSketch.θ ← ε ‖A_t‖_F^2
    AeroSketch.Update(a_i)
Function Query(t):
    return AeroSketch.Query(0, t)
```

Theorem 4.1. *Algorithm 8 solves the ATTP Matrix Sketch (Problem 3) with space complexity* $O\left(\frac{d}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$ *and amortized time complexity* $O\left(\frac{d}{\varepsilon}\log d\right)$ *per update, with success probability at least that given in Eq.* (2). *Alternatively, using the probability amplification of Algorithm 7, it achieves amortized time complexity* $O\left(\frac{d}{\varepsilon}\log d\log^2\frac{1}{\delta}\right)$ *per update while guarantee success probability at least* $1-\delta$, *where* δ *is a tunable parameter that can be chosen arbitrarily in the interval* $(0, 1/100)$.

The detailed proof can be found in our technical report [1].

4.2 Tracking Approximate Matrix Sketch over Sliding Window (Problem 4)

Algorithm 9 shows the sketching algorithm for tracking an approximate matrix sketch over sliding windows. The high-level idea and algorithm skeleton are similar to hDS-COD (Algorithm 3 in [42]) and SO-COD (Algorithm 2 in [40]), but we replace the DS-COD subroutines with AeroSketchCOD to avoid frequent and time-consuming matrix factorizations.

Algorithm 9: AeroSketchCOD Framework for Problem 4

Function Initialize($d_x, d_y, \varepsilon, N, \theta$):
  Input: d_x, d_y: dimension of the vectors in stream; N: sliding window size; ε: relative covariance error bound; θ: dump threshold.
  $\ell \leftarrow \lceil \frac{2}{\varepsilon} \rceil, \quad \hat{\boldsymbol{A}} \leftarrow \boldsymbol{0}^{d_x \times l}, \quad \hat{\boldsymbol{B}} \leftarrow \boldsymbol{0}^{d_y \times l}$
  $\mathcal{S} \leftarrow$ an empty queue
Function Update($\boldsymbol{x}_i, \boldsymbol{y}_i$):
  Input: $\boldsymbol{x}_i$: the column vector of $\boldsymbol{X}$ arriving at time i; $\boldsymbol{y}_i$: the column vector of $\boldsymbol{Y}$ arriving at time i;
  while $\mathcal{S}[0].t + N \leq i$ **do** // snapshot expired
    $\mathcal{S}$.popleft()
  $\tilde{\boldsymbol{A}} \leftarrow \begin{bmatrix} \boldsymbol{A}_{i-1}^\top & \boldsymbol{a}_i \end{bmatrix}^\top, \tilde{\boldsymbol{B}} \leftarrow \begin{bmatrix} \boldsymbol{B}_{i-1}^\top & \boldsymbol{b}_i \end{bmatrix}^\top$
  if $rows(\tilde{\boldsymbol{A}}) \geq 2\ell$ **then** // $\tilde{\boldsymbol{A}}$ has no zero rows
    $\boldsymbol{A}'_i, \boldsymbol{B}'_i \leftarrow \mathrm{COD}(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}, \ell)$ // Co-Occuring Directions Sketch, refer to [25]
  else
    $\boldsymbol{A}'_i, \boldsymbol{B}'_i \leftarrow \tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}$
  $\hat{\sigma}_1^2, \hat{\boldsymbol{v}}_1 \leftarrow \mathtt{power\_iteration}(\boldsymbol{A}'_i \boldsymbol{B}'^\top_i, k = O(\log_2 d))$
  if $\hat{\sigma}_1 \geq \frac{\theta}{2}$ **then** // largest singular value
    for $j = 1$ **to** $\lceil \log_2 2\ell \rceil$ **do**
      $k \leftarrow \min(2^j, 2\ell)$
      $[\boldsymbol{Z}', \hat{\Sigma}] \leftarrow \mathtt{simul\_iter}(\boldsymbol{A}'_i \boldsymbol{B}'^\top_i, k, \varepsilon_{\mathrm{SI}})$
      $[\boldsymbol{H}', \hat{\Sigma}] \leftarrow \mathtt{simul\_iter}(\boldsymbol{B}'_i \boldsymbol{A}'^\top_i, k, \varepsilon_{\mathrm{SI}})$
      if $\hat{\Sigma}_k < \theta$ **then**
        $k' \leftarrow \arg\max_{k'} \{\hat{\Sigma}_{k'} \geq \theta\}$
        $\boldsymbol{Z}_i \leftarrow \boldsymbol{Z}'[:, 1:k'], \quad \boldsymbol{H}_i \leftarrow \boldsymbol{H}'[:, 1:k']$
        $\mathtt{snapshot} \leftarrow (\boldsymbol{Z}_i, \boldsymbol{Z}_i^\top \boldsymbol{A}'_i \boldsymbol{B}'^\top_i, \boldsymbol{A}'_i \boldsymbol{B}'^\top_i \boldsymbol{H}_i^\top, \boldsymbol{H}_i)$
        $\mathcal{S}$.append(snapshot, $s = \mathcal{S}[-1].t + 1, t = i$)
        $\boldsymbol{A}_i \leftarrow (\boldsymbol{I} - \boldsymbol{Z}\boldsymbol{Z}^\top)\, \boldsymbol{A}'_i, \ \boldsymbol{B}_i \leftarrow (\boldsymbol{I} - \boldsymbol{H}\boldsymbol{H}^\top)\, \boldsymbol{B}'_i$
        break
  else
    $\boldsymbol{A}_i \leftarrow \boldsymbol{A}'_i, \ \boldsymbol{B}_i \leftarrow \boldsymbol{B}'_i$
Function Query():
  $\boldsymbol{A}\boldsymbol{B}^\top \leftarrow \boldsymbol{A}_T \boldsymbol{B}_T^\top$
  forall $(\boldsymbol{Z}_i, \boldsymbol{Z}_i^\top \boldsymbol{A}'_i \boldsymbol{B}'^\top_i, \boldsymbol{A}'_i \boldsymbol{B}'^\top_i \boldsymbol{H}_i^\top, \boldsymbol{H}_i)$ *in* $\mathcal{S}$ **do**
    $\boldsymbol{A}\boldsymbol{B}^\top \leftarrow \boldsymbol{A}\boldsymbol{B}^\top + \boldsymbol{Z}_i \boldsymbol{Z}_i^\top \boldsymbol{A}'_i \boldsymbol{B}'^\top_i + \boldsymbol{A}'_i \boldsymbol{B}'^\top_i \boldsymbol{H}_i^\top \boldsymbol{H}_i - \boldsymbol{Z}_i \boldsymbol{Z}_i^\top \boldsymbol{A}'_i \boldsymbol{B}'^\top_i \boldsymbol{H}_i^\top \boldsymbol{H}_i$
  $[\boldsymbol{U}, \Sigma, \boldsymbol{V}^\top] \leftarrow \mathrm{SVD}(\boldsymbol{A}\boldsymbol{B}^\top)$
  $\boldsymbol{A}^* \leftarrow \boldsymbol{U}\sqrt{\max(\Sigma - \sigma_\ell \boldsymbol{I}, \boldsymbol{0})}, \quad \boldsymbol{B}^* \leftarrow \boldsymbol{V}\sqrt{\max(\Sigma - \sigma_\ell \boldsymbol{I}, \boldsymbol{0})}$
  return $\boldsymbol{A}^*[:, 1:\ell], \quad \boldsymbol{B}^*[:, 1:\ell]$

Theorem 4.2. *Algorithm 9 solves the Tracking Approximate Matrix Multiplication over Sliding Window (Problem 4) with space complexity* $O\left(\frac{d_x+d_y}{\varepsilon}\log R\right)$ *and amortized time complexity* $O\left(\frac{d_x+d_y}{\varepsilon}\log d\log R\right)$ *per update, with success probability at least that given in Eq.* (2). *Alternatively, using the similar like probability amplification of Algorithm 7, it achieves amortized time complexity* $O\left(\frac{d_x+d_y}{\varepsilon}\log d\log R\log^2\frac{1}{\delta}\right)$ *per update while guarantee success probability at least* $1-\delta$, *where* δ *is a tunable parameter that can be chosen arbitrarily in the interval* $(0, 1/100)$.

The detailed proof can be found in our technical report [1].

4.3 Distributed Matrix Sketch (Problem 5)

Algorithm 10 describes how AeroSketch helps to solve the Tracking Distributed Matrix Sketch problem. The high-level idea and algorithm skeleton are similar to P2 (Algorithms 5.3 and 5.4 in [10]), although we replace the SVD with AeroSketch to avoid frequent and time-consuming matrix factorizations.

Algorithm 10: P5: AeroSketch Framework for Problem 5

Procedure Update($\boldsymbol{a}_i$): // at Site j
  $F_j \leftarrow F_j + \|\boldsymbol{a}_i\|_2^2$
  if $F_j \geq \frac{\varepsilon}{m}\hat{F}$ **then**
    Send F_j to coordinator; set $F_j \leftarrow 0$
  AeroSketch.Update($\boldsymbol{a}_i$)
  if *AeroSketch*.$\mathcal{S}$ *is not empty* **then**
    Send $\boldsymbol{Z}, \boldsymbol{Z}^\top\boldsymbol{C}^\top\boldsymbol{C}$ of $\mathcal{S}[0]$ to coordinator
    Delete $\mathcal{S}[0]$ then AeroSketch.$\mathcal{S}$ is empty
    return $\boldsymbol{Z}, \boldsymbol{Z}^\top\boldsymbol{C}^\top\boldsymbol{C}$
  case *received* $\hat{F}$ **do**
    AeroSketch.$\theta \leftarrow \varepsilon\hat{F}$
Procedure Receive_Update: // at Coordinator
  case *received* F_j **do**
    $\hat{F}$ += F_j and `#msg` += 1
    if `#msg` $\geq m$ **then**
      `#msg` $\leftarrow$ 0, then Broadcast $\hat{F}$ to all sites
  case *received* $\boldsymbol{Z}$ *and* $\boldsymbol{Z}^\top\boldsymbol{C}^\top\boldsymbol{C}$ **do**
    $\boldsymbol{B}$ += $\boldsymbol{Z}\boldsymbol{Z}^\top\boldsymbol{C}^\top\boldsymbol{C} + \boldsymbol{C}^\top\boldsymbol{C}\boldsymbol{Z}\boldsymbol{Z}^\top - \boldsymbol{Z}\boldsymbol{Z}^\top\boldsymbol{C}^\top\boldsymbol{C}\boldsymbol{Z}\boldsymbol{Z}^\top$
Procedure Query: // at Coordinator
  $[\boldsymbol{V}, \boldsymbol{\Lambda}, \boldsymbol{V}^\top] \leftarrow$ `eigen_decomposition`($\boldsymbol{B}$)
  $\hat{\boldsymbol{B}} \leftarrow \sqrt{\max\left(\boldsymbol{\Lambda} - \boldsymbol{I}\lambda_\ell, \boldsymbol{0}\right)}\,\boldsymbol{V}^\top$
  return $\hat{\boldsymbol{B}}[1:\ell, :]$

Theorem 4.3. *P5 (Algorithm 10) solves the Distributed Matrix Sketch problem (Problem 5) with communication complexity* $O\left(\frac{md}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$ *and amortized time complexity* $O\left(\frac{d}{\varepsilon}\log d\right)$ *per update, with success probability at least that given in Eq.* (2). *Alternatively, using the probability amplification*

of Algorithm 7, it achieves amortized time complexity $O\left(\frac{d}{\varepsilon}\log d\log^2\frac{1}{\delta}\right)$ *per update while guarantee success probability at least* $1-\delta$*, where* δ *is a tunable parameter that can be chosen in the interval* $(0, 1/100)$.

The detailed proof can be found in our technical report [1].

4.3.1 Tracking Distributed Matrix Sketch over Sliding Window (Problem 6). Algorithm 11 describes how P5 (Algorithm 10) can be used to solve the Tracking Distributed Matrix Sketch over Sliding Window problem. The high-level idea and algorithm skeleton are similar to DA2 (Algorithm 5 in [45]). The IWMT protocol used by DA2 is a "black-box" one-way communication version of P2. Here we replace IWMT with P5 (Algorithm 10) to achieve the same communication complexity while avoiding frequent and time-consuming matrix factorizations.

Algorithm 11: AeroSketch Framework for Problem 6

Procedure Update($\boldsymbol{a}_i$): // at Site j
  $Z, Z^\top C^\top C \leftarrow$ P5.Update($\boldsymbol{a}_i$)
  Send $(Z, Z^\top C^\top C, \texttt{flag} = \texttt{update})$
  mEH_a.Update($\boldsymbol{a}_i$);
  if $i \mod N \equiv 0$ **then**
    $Q \leftarrow IWMT_c(mEH_a)$
  for $\boldsymbol{e}_t \in Q$ *and* $t < i - N$ **do**
    Send $(\boldsymbol{e}_t, \texttt{flag} = \texttt{expire})$, then remove $\boldsymbol{e}_t$ from Q
Procedure Receive_Update: // at Coordinator
  switch flag **do**
    case update **do**
      $\boldsymbol{B} \mathrel{+}= ZZ^\top C^\top C + C^\top CZZ^\top - ZZ^\top C^\top CZZ^\top$
    case expire **do**
      $\boldsymbol{B} \mathrel{-}= \boldsymbol{e}_t^\top \boldsymbol{e}_t$
Procedure Query: // at Coordinator
  $[\boldsymbol{V}, \boldsymbol{\Lambda}, \boldsymbol{V}^\top] \leftarrow \texttt{eigen\_decomposition}(\boldsymbol{B})$
  $\hat{\boldsymbol{B}} \leftarrow \sqrt{\max\left(\boldsymbol{\Lambda} - \boldsymbol{I}\lambda_\ell, \boldsymbol{0}\right)}\,\boldsymbol{V}^\top$
  return $\hat{\boldsymbol{B}}[1:\ell, :]$

Theorem 4.4. *Algorithm 11 solves the Tracking Distributed Matrix Sketch over Sliding Window (Problem 6) with communication complexity* $O\left(\frac{md}{\varepsilon}\log\|\boldsymbol{A}\|_F^2\right)$ *and amortized time complexity* $O\left(\frac{d}{\varepsilon}\log d\right)$ *per update, with success probability at least that given in Eq.* (2)*. Alternatively, using the probability amplification of Algorithm 7, it achieves amortized time complexity* $O\left(\frac{d}{\varepsilon}\log d\log^2\frac{1}{\delta}\right)$ *per update to guarantee success probability at least* $1-\delta$*, where* δ *is a tunable parameter that can be chosen arbitrarily in the interval* $(0, 1/100)$.

The detailed proof can be found in our technical report [1].

5 Experiments

5.1 Experimental Setup

5.1.1 Problem Scenarios, Algorithms and Parameter Settings. We evaluate our proposed framework AeroSketch against the space-/communication-optimal baseline algorithms listed in Table 1 for various tracking matrix sketching problems:

- **Matrix Sketch over Sliding Window (Problem 2)**: Fast-DS-FD [44].
- **ATTP Matrix Sketch (Problem 3)**: PFD [30].
- **Tracking Approximate Matrix Multiplication over Sliding Window (AMM, Problem 4)**: hDS-COD/SO-COD [40, 42].
- **Tracking Distributed Matrix Sketch (Problem 5)**: P2 [10].
- **Distributed Matrix Sketch over Sliding Window (Distributed SW, Problem 6)**: DA2 [45].

The **bold-underlined** terms are the abbreviations used in the experimental result figures. For all algorithms, common parameters are the dimension of the vector stream, denoted as d, and the upper bound on the relative covariance error, ε. Our experiments primarily focus on comparing various performance metrics of AeroSketch against baseline algorithms under varying values of ε. The specific metrics we focus on will be elaborated in Section 5.1.2.

5.1.2 Metrics.

- **Maximum sketch size** assesses the memory cost of a matrix sketch algorithm. For each algorithm, we record the peak total memory usage of all data structures (in number of rows or KB) during runtime over the entire stream.
- **Empirical relative covariance error** assesses the quality of the approximate sketch matrix produced by algorithms. It is defined as $\|\boldsymbol{A}^\top\boldsymbol{A} - \boldsymbol{B}^\top\boldsymbol{B}\|_2/\|\boldsymbol{A}\|_F^2$, where $\boldsymbol{A}$ is the ground-truth matrix and $\boldsymbol{B}$ is the sketch matrix generated by the algorithm. In the case of approximate matrix multiplication, the relative error is defined as $\|\boldsymbol{X}\boldsymbol{Y}^\top - \boldsymbol{A}\boldsymbol{B}^\top\|_2/(\|\boldsymbol{X}\|_F\|\boldsymbol{Y}\|_F)$, where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the input stream matrices, while $\boldsymbol{A}$ and $\boldsymbol{B}$ are the resulting sketches. It is important to note that the parameter ε is an upper bound on the relative covariance error (i.e., the worst-case approximation guarantee). In practice, the actual relative covariance error observed during experiments may be significantly lower than ε. Even under the same ε setting, different algorithms may exhibit different empirically measured covariance errors.
- **Amortized update time** refers to the average time to update for each input vector from the streams during the runtime.
- **Speedup ratio** measures how much faster AeroSketch is compared with the baseline. It is calculated as the ratio of the amortized update time of the baseline algorithm to the amortized update time of the AeroSketch algorithm.
- **Communication cost**: In distributed settings, this metric captures the total size of messages exchanged between sites and the coordinator during the execution of the sketch algorithm.
- **Space/communication cost ratio** compares the space/communication footprint of AeroSketch algorithm with the baseline algorithm, calculated as the ratio of space/communication cost of our AeroSketch algorithms to that of the baseline algorithms.

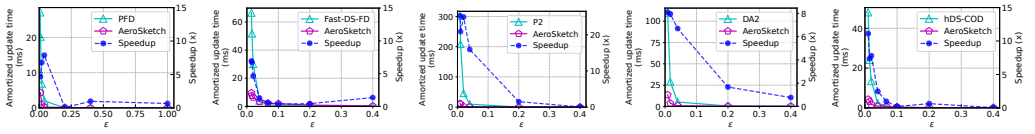
5.1.3 Hardware, Environments, and Datasets. All algorithms are implemented in Python 3.12.0. To ensure stable timing measurements, experiments are conducted on a single idle core of an Intel® Xeon® CPU E7-4809 v4 clocked at 2.10 GHz. Each algorithm is run on each dataset 20 times, and the average of the update time is recorded. For probabilistic algorithms such as Power Iteration and Simultaneous Iteration, we use a fixed random seed to ensure the reproducibility of our results. The

experiments conducted in distributed settings are implemented with the help of Ray[3] 2.46.0 [24]. We query and record the sketch and metrics every 20 steps. The datasets we use are:

(1) **Uniform Random**: The entries of matrices are drawn uniformly at random from the interval $(0, 1]$. We use this dataset for AMM. We set $d_x = 300, d_y = 500, N = 5000, T = 10000$.
(2) **Random Noisy** [33]: The matrices are generated by the formula $\boldsymbol{A} = \boldsymbol{SDU} + \boldsymbol{N}/\zeta$. Here, $\boldsymbol{S}$ is an $n \times d$ matrix of signal coefficients, with each entry drawn from a standard normal distribution. $\boldsymbol{D}$ is a diagonal matrix with $\boldsymbol{D}_{i,i} = 1 - (i-1)/d$. $\boldsymbol{U}$ represents the signal row space, satisfying $\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}_d$, and ζ is a positive real parameter for adjusting noise intensity to control the signal-to-noise ratio. The matrix $\boldsymbol{N}$ adds Gaussian noise, with $\boldsymbol{N}_{i,j}$ drawn from the standard Gaussian distribution $\mathcal{N}(0, 1)$. We use the dataset for ATTP, Distributed (SW). For ATTP, we set $d = 500, T = 10000$. For Distributed, we set $m = 4, d = 500, T = 10000$. For Distributed (SW), we set $m = 4, d = 500, N = 5000, T = 10000$.
(3) **Real-world Dataset**: GloVe[4] provides a dataset of pre-trained word vectors [29] with dimension of $d = 300$. We use the dataset for the Sliding Window. We set $N = 5000, T = 10000$.

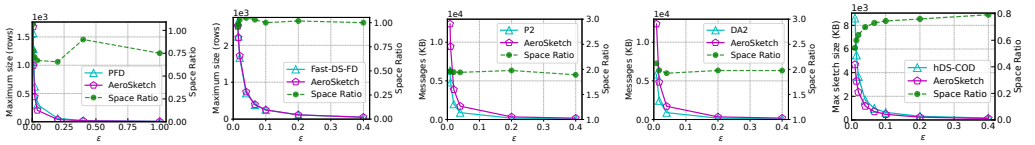
5.2 Experimental Results

Figures 3-10 show the comparison of different metrics between the baseline algorithms and the AeroSketch algorithms across the five problem scenarios. Among them, the cyan lines represent the space/communication-optimal baseline algorithms, while the magenta lines represent the AeroSketch-optimized algorithms proposed in this work. The blue lines represent the speedup ratio, while the green lines represent the space/communication cost ratio.



(a) ATTP (b) Sliding Window (c) Distributed (d) Distributed SW (e) AMM

Fig. 3. Amortized update time vs. parameter ε.



(a) ATTP (b) Sliding Window (c) Distributed (d) Distributed SW (e) AMM

Fig. 4. Space/communication cost vs. parameter ε.

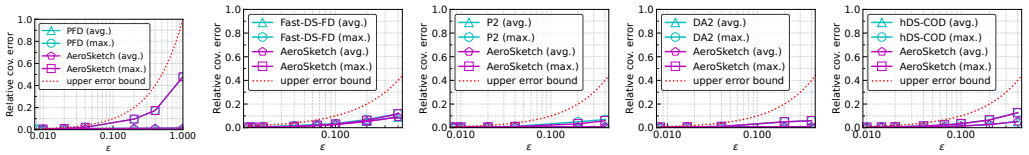
5.2.1 ε vs. Time, Space, and Communication Cost. For both the baseline algorithm and our AeroSketch algorithm, we set a series of parameters ε (corresponding to the x-axis in Figures 3 and 4) for each of the five problem scenarios, and recorded the amortized update time and maximum space cost (or communication cost under the distributed scenario) of both algorithms under each ε setting, corresponding to the y-axis in Figures 3 and 4.

Figure 3 shows the relationship between the parameter ε and the amortized update time. When ε is set to a larger value, our AeroSketch algorithms show insignificant acceleration compared to the baseline algorithms. However, as ε gradually converges to 0, the speedup ratio of our

[3]https://www.ray.io/
[4]https://nlp.stanford.edu/projects/glove/

AeroSketch algorithms relative to the baseline algorithms increases significantly, which aligns with the conclusions in Table 1. Taking Figure 3b as an example, the time complexity corresponding to Fast-DS-FD is $O\left(\left(\frac{d}{\varepsilon}+\frac{1}{\varepsilon^3}\right)\log R\right)$, while the time complexity of our proposed AeroSketch algorithms in Theorem 3.1 of this paper is $O\left(\frac{d}{\varepsilon}\log d\log R\right)$. When $\varepsilon > 1/\sqrt{d}$ (thus $d/\varepsilon > 1/\varepsilon^3$), the $1/\varepsilon^3$ term in the baseline algorithm is negligible compared to d/ε. However, as ε starts from $\varepsilon = O(1/d)$ and gradually converges to 0, the $1/\varepsilon^3$ term in the baseline algorithm's time complexity begins to dominate, causing the overall time complexity to degrade to a cubic order of $O(d^3)$. In contrast, our AeroSketch achieves a near-quadratic order of $O(d^2 \log d)$. Similar conclusions can be drawn for other scenarios in Table 1 using analogous analysis, indicating that our AeroSketch can prevent performance degradation as ε converges to 0, ensuring better time performance even under stricter error bounds.

Figure 4 shows the relationship between the space/communication cost of the algorithms and parameter ε. We observe that across all values of ε in each problem scenario, the space/communication cost ratio between our AeroSketch algorithm and the baseline algorithm stabilizes at a constant between 0.5 and 2. This aligns with the conclusion in Table 1 that both the AeroSketch algorithm and the baseline algorithm achieve the same order and optimal asymptotic space/communication complexity. Theoretically, under the same average error, the space or communication cost of AeroSketch is approximately twice that of the previous state-of-the-art algorithm, as shown in Figures 4c and 4d, as the communication cost ratio is close to 2. The reason, as noted earlier, is that the baseline algorithm computes snapshots exactly, while our approximate algorithm AeroSketch must store an additional subspace-generating matrix Z_i (line 15 in Algorithm 4) to recover the original matrix from the residual matrix C_i, thus introducing a constant factor of 2. However, we observe that certain space optimization techniques can be employed to reduce the constant factors. For example, in PFD, recording partial checkpoints and full checkpoints can be optimized to record a single type of snapshot; the auxiliary queue in Fast-DS-FD and hDS-COD/SO-COD has been proven to be removable. These techniques can make the actual space/communication cost of AeroSketch essentially equal to or even slightly lower than that of the baseline algorithm under the same relative error upper bound ε, as reflected in Figures 4a, 4b, and 4e, where the space cost ratio is equal to or slightly below 1. Overall, both theoretical analysis and experimental results verify that AeroSketch achieves the same order-optimal asymptotic space complexity as the baseline.

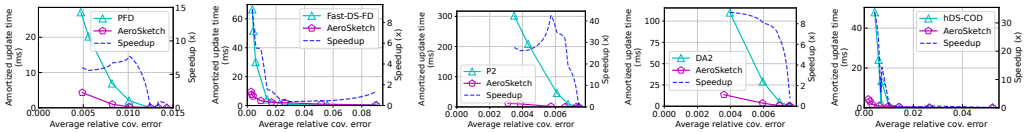


(a) ATTP (b) Sliding Window (c) Distributed (d) Distributed SW (e) AMM

Fig. 5. Empirical relative covariance errors vs. parameter ε.

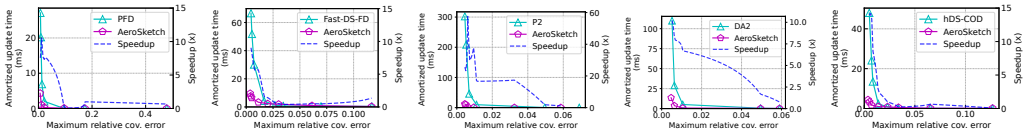
Remark. *It is important to note that the parameter ε used in the algorithm is an upper bound on the empirical relative covariance error (i.e., ε is the worst-case approximation guarantee). As shown in Figure 5, the maximum empirical relative covariance error of all algorithms is lower than the upper bound parameter ε, which validates the correctness of our algorithm implementation. Moreover, even under the same ε setting, different algorithms may exhibit different empirically measured covariance errors. Given this, beyond the analysis of ε's effect on performance metrics of time and space/communication cost in the previous subsection, we now turn to a comparison of the time and*

*space/communication cost of different algorithms under the same **empirical errors** in the next subsection.*



(a) ATTP (b) Sliding Window (c) Distributed (d) Distributed SW (e) AMM

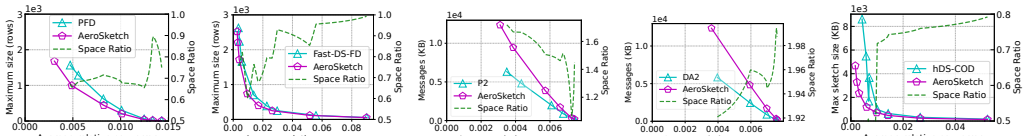
Fig. 6. Amortized update time vs. Average error



(a) ATTP (b) Sliding Window (c) Distributed (d) Distributed SW (e) AMM

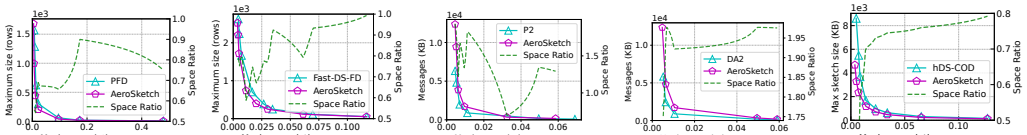
Fig. 7. Amortized update time vs. Maximum error

5.2.2 Empirical Error vs. Time vs. Space/Communication Cost. Figures 6 and 7 show the trade-off between amortized update time and avg./max. relative error across the five problem scenarios. We observe that, under the same relative error guarantee (i.e., along the same x-axis), the amortized update time of the AeroSketch algorithm is generally lower than that of the baseline algorithms. When the empirical error is not very small, the time gap between the two algorithms is limited. However, as the error bound tightens, the time advantage of the AeroSketch optimization becomes more pronounced. This aligns with our theoretical analysis: as ε converges to 0, the update time complexity of baseline algorithms degrades to $O(d^3)$, while AeroSketch maintains $O(d^2 \log d)$.



(a) ATTP (b) Sliding Window (c) Distributed (d) Distributed SW (e) AMM

Fig. 8. Space/communication cost vs. Average error

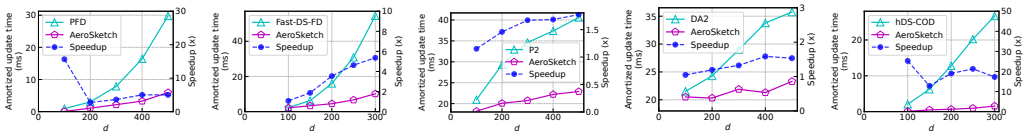


(a) ATTP (b) Sliding Window (c) Distributed (d) Distributed SW (e) AMM

Fig. 9. Space/communication cost vs. Maximum error

Figures 8 and 9 show the space/communication costs vs. error. We observe that, under the same empirical error, the space/communication costs of baseline algorithms and our AeroSketch are

comparable. In some cases, AeroSketch even achieves lower space costs, consistent with our earlier theoretical analysis that its space/communication complexity is of the same order and asymptotically optimal compared to baseline algorithms.



(a) ATTP (b) Sliding Window (c) Distributed (d) Distributed SW (e) AMM

Fig. 10. d vs. Amortized update time

5.2.3 Asymptotic analysis of amortized update time complexity with respect to dimension d. In addition, the time efficiency of AeroSketch becomes more prominent for higher-dimensional vector streams. As we analyzed earlier, when ε tightens, the update time complexities of previous baseline algorithms degrade to a cubic order of $O(d^3)$. In contrast, our AeroSketch achieves a near-quadratic order of $O(d^2 \log d)$. The relationship between the empirical amortized update time and the vector dimension d plotted in Figure 10 clearly validates this point. For each value of d, we set $\varepsilon = 4/d = O(1/d)$. We observe that as d increases, the time costs of baseline algorithms grow significantly faster compared to AeroSketch. As dimension d increases further, the time advantage of AeroSketch is expected to become even more substantial.

6 Conclusion and Future Work

In this paper, we present a new framework, AeroSketch, for optimizing the update time complexity of matrix sketch algorithms to a near-optimal level, up to a logarithmic factor. We show that AeroSketch can address a wide range of matrix sketch problems while achieving optimal asymptotic space complexity. We provide end-to-end proofs for the error bounds, time complexity, and space complexity to demonstrate the correctness and effectiveness of our approach. We implement the algorithms optimized by our framework and evaluate them on various matrix sketch problems, using large-scale synthetic and real-world streams, in comparison with baseline algorithms. The results show that our framework significantly outperforms prior methods in terms of update time, especially under tight approximation error constraints. Meanwhile, it remains competitive in space and communication cost. The experimental results corroborate our theoretical analysis.

Potential future work encompasses the following direction: Is there a (truly deterministic) algorithm that achieves a truly optimal amortized update time complexity $\Omega(d/\varepsilon)$, rather than the near-optimal one with a logarithmic factor—thus outperforming AeroSketch?

Acknowledgments

This research was supported in part by National Science and Technology Major Project (2022ZD0114802), by National Natural Science Foundation of China (No. 92470128, No. U2241212). We also wish to acknowledge the support provided by the fund for building world-class universities (disciplines) of Renmin University of China, by Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education, by Intelligent Social Governance Interdisciplinary Platform, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, Public Policy and Decision-making Research Lab, and Public Computing Cloud, Renmin University of China.

We thank Mingji Yang and Guanyu Cui for their valuable comments that improved the presentation of this article. We also thank Professor Sibo Wang for his many helpful suggestions.

References

[1] 2026. Technical Report. https://arxiv.org/abs/2601.02019

[2] Andre L. L. Aquino, Carlos Mauricio S. Figueiredo, Eduardo Freire Nakamura, Luciana S. Buriol, Antonio Alfredo Ferreira Loureiro, Antônio Otávio Fernandes, and Claudionor José Nunes Coelho. 2007. A Sampling Data Stream Algorithm For Wireless Sensor Networks. *2007 IEEE International Conference on Communications* (2007), 3207–3212. https://api.semanticscholar.org/CorpusID:14878450

[3] Cheng Chen, Luo Luo, Weinan Zhang, Yong Yu, and Yijiang Lian. 2020. Efficient and Robust High-Dimensional Linear Contextual Bandits.. In *IJCAI*. 4259–4265.

[4] Cheng Chen, Weinan Zhang, and Yong Yu. 2022. Efficient policy evaluation by matrix sketching. *Frontiers of Computer Science* 16, 5 (2022), 165330.

[5] Graham Cormode and Ke Yi. 2020. *Small summaries for big data.* Cambridge University Press.

[6] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining stream statistics over sliding windows. *SIAM journal on computing* 31, 6 (2002), 1794–1813.

[7] Amey Desai, Mina Ghashami, and Jeff M Phillips. 2016. Improved practical matrix sketching with guarantees. *IEEE Transactions on Knowledge and Data Engineering* 28, 7 (2016), 1678–1690.

[8] Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. 2016. Frequent directions: Simple and deterministic matrix sketching. *SIAM J. Comput.* 45, 5 (2016), 1762–1792.

[9] Mina Ghashami and Jeff M Phillips. 2014. Relative errors for deterministic low-rank matrix approximations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 707–717.

[10] Mina Ghashami, Jeff M. Phillips, and Feifei Li. 2014. Continuous Matrix Approximation on Distributed Data. *Proc. VLDB Endow.* 7, 10 (2014), 809–820. doi:10.14778/2732951.2732954

[11] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.

[12] Zengfeng Huang. 2019. Near optimal frequent directions for sketching dense and sparse matrices. *Journal of Machine Learning Research* 20, 56 (2019), 1–23.

[13] Zengfeng Huang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2017. Efficient matrix sketching over distributed data. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 347–359.

[14] Zengfeng Huang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2021. Communication-efficient distributed covariance sketch, with application to distributed PCA. *Journal of Machine Learning Research* 22, 80 (2021), 1–38.

[15] Praneeth Kacham. 2024. *On Efficient Sketching Algorithms.* Ph. D. Dissertation. Carnegie Mellon University.

[16] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. 2020. Tea: Enabling state-intensive network functions on programmable switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 90–106.

[17] Jacek Kuczyński and Henryk Woźniakowski. 1992. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. *SIAM journal on matrix analysis and applications* 13, 4 (1992), 1094–1122.

[18] Ilja Kuzborskij, Leonardo Cella, and Nicolò Cesa-Bianchi. 2019. Efficient linear bandits through matrix sketching. In *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 177–185.

[19] Lap-Kei Lee and HF Ting. 2006. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 290–297.

[20] Edo Liberty. 2013. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 581–588.

[21] Edo Liberty. 2022. Even simpler deterministic matrix sketching. *arXiv preprint arXiv:2202.01780* (2022).

[22] Per-Gunnar Martinsson. 2016. Randomized methods for matrix computations. *IAS/Park City Mathematics Series* (2016). https://api.semanticscholar.org/CorpusID:56112037

[23] Per-Gunnar Martinsson and Joel A Tropp. 2020. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica* 29 (2020), 403–572.

[24] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, 561–577. https://www.usenix.org/conference/osdi18/presentation/nishihara

[25] Youssef Mroueh, Etienne Marcheret, and Vaibahava Goel. 2017. Co-occurring directions sketching for approximate matrix multiply. In *Artificial Intelligence and Statistics*. PMLR, 567–575.

[26] Riley Murray, James Demmel, Michael W. Mahoney, N. Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michal Derezinski, Miles E. Lopes, Tianyu Liang, Hengrui Luo, and Jack J. Dongarra.

2023. Randomized Numerical Linear Algebra : A Perspective on the Field With an Eye to Software. *CoRR* abs/2302.11474 (2023). arXiv:2302.11474 doi:10.48550/ARXIV.2302.11474

[27] Cameron Musco and Christopher Musco. 2015. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Advances in neural information processing systems* 28 (2015).

[28] S. Muthukrishnan. 2005. Data Streams: Algorithms and Applications. *Found. Trends Theor. Comput. Sci.* 1, 2 (2005). doi:10.1561/0400000002

[29] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[30] Benwei Shi, Zhuoyue Zhao, Yanqing Peng, Feifei Li, and Jeff M Phillips. 2021. At-the-time and back-in-time persistent sketches. In *Proceedings of the 2021 International Conference on Management of Data*. 1623–1636.

[31] Lloyd N Trefethen and David Bau. 2022. *Numerical linear algebra*. SIAM.

[32] Joel A Tropp and Robert J Webber. 2023. Randomized algorithms for low-rank matrix approximation: Design, analysis, and applications. *arXiv preprint arXiv:2306.12418* (2023).

[33] Roman Vershynin. 2011. Spectral norm of products of random and deterministic matrices. *Probability theory and related fields* 150, 3 (2011), 471–509.

[34] Jing Wang, Miao Yu, Peng Zhao, and Zhi-Hua Zhou. 2024. Learning with Adaptive Resource Allocation. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. 52099–52116.

[35] Zhewei Wei, Xuancheng Liu, Feifei Li, Shuo Shang, Xiaoyong Du, and Ji-Rong Wen. 2016. Matrix sketching over sliding windows. In *Proceedings of the 2016 International Conference on Management of Data*. 1465–1480.

[36] Dongxie Wen, Hanyan Yin, Xiao Zhang, and Zhewei Wei. 2024. Matrix Sketching in Bandits: Current Pitfalls and New Framework. *arXiv preprint arXiv:2410.10258* (2024).

[37] Dongxie Wen, Xiao Zhang, Zhewei Wei, Chenping Hou, Shuai Li, and Weinan Zhang. 2025. Fast Second-Order Online Kernel Learning Through Incremental Matrix Sketching and Decomposition. (8 2025), 6552–6560. doi:10.24963/ijcai.2025/729 Main Track.

[38] David P. Woodruff. 2014. Sketching as a Tool for Numerical Linear Algebra. *Found. Trends Theor. Comput. Sci.* 10, 1-2 (2014), 1–157. doi:10.1561/0400000060

[39] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. 2014. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials* 17, 1 (2014), 27–51.

[40] Haoming Xian, Qintian Guo, Jun Zhang, and Sibo Wang. 2025. Optimal Approximate Matrix Multiplication over Sliding Window. *Proc. VLDB Endow.* 19, 3 (2025), 455–467. doi:10.14778/3778092.3778105

[41] Rui Xie, Shuyang Bai, and Ping Ma. 2023. Optimal sampling designs for multidimensional streaming time series with application to power grid sensor data. *The Annals of applied statistics* 17, 4 (2023), 3195–3215.

[42] Ziqi Yao, Mingsong Chen, and Cheng Chen. 2025. Optimal Approximate Matrix Multiplication over Sliding Windows. *arXiv preprint arXiv:2502.18830* (2025).

[43] Ziqi Yao, Lianzhi Li, Mingsong Chen, Xian Wei, and Cheng Chen. 2024. Approximate Matrix Multiplication over Sliding Windows. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3896–3906.

[44] Hanyan Yin, Dongxie Wen, Jiajun Li, Zhewei Wei, Xiao Zhang, Zengfeng Huang, and Feifei Li. 2024. Optimal Matrix Sketching over Sliding Windows. *Proc. VLDB Endow.* 17, 9 (May 2024), 2149–2161. doi:10.14778/3665844.3665847

[45] Haida Zhang, Zengfeng Huang, Zhewei Wei, Wenjie Zhang, and Xuemin Lin. 2017. Tracking matrix approximation over distributed sliding windows. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 833–844.

[46] Zhi-Hua Zhou. 2024. Learnability with time-sharing computational resource concerns. *National Science Review* 11, 10 (2024), nwae204.

Received July 2025; revised October 2025; accepted November 2025