Optimal Matrix Sketching over Sliding Windows

Hanyan Yin
Renmin University of China
yinhanyan@ruc.edu.cn

Dongxie Wen
Renmin University of China
2019202221@ruc.edu.cn

Jiajun Li
Renmin University of China
2015201613@ruc.edu.cn

Zhewei Wei*
Renmin University of China
zhewei@ruc.edu.cn

Xiao Zhang
Renmin University of China
zhangx89@ruc.edu.cn

Zengfeng Huang
Fudan University
huangzf@fudan.edu.cn

Feifei Li
Alibaba Group
lifeifei@alibaba-inc.com

ABSTRACT

Matrix sketching, aimed at approximating a matrix $\boldsymbol{A} \in \mathbb{R}^{N\times d}$ consisting of vector streams of length N with a smaller sketching matrix $\boldsymbol{B} \in \mathbb{R}^{\ell\times d}$, $\ell \ll N$, has garnered increasing attention in fields such as large-scale data analytics and machine learning. A well-known deterministic matrix sketching method is the FrequentDirections algorithm, which achieves the optimal $O\left(\frac{d}{\varepsilon}\right)$ space bound and provides a covariance error guarantee of $\varepsilon = \|\boldsymbol{A}^\top\boldsymbol{A} - \boldsymbol{B}^\top\boldsymbol{B}\|_2/\|\boldsymbol{A}\|_F^2$. The matrix sketching problem becomes particularly interesting in the context of sliding windows, where the goal is to approximate the matrix $\boldsymbol{A}_W$, formed by input vectors over the most recent N time units. However, despite recent efforts, whether achieving the optimal $O\left(\frac{d}{\varepsilon}\right)$ space bound on sliding windows is possible has remained an open question.

In this paper, we introduce the DS-FD algorithm, which achieves the optimal $O\left(\frac{d}{\varepsilon}\right)$ space bound for matrix sketching over row-normalized, sequence-based sliding windows. We also present matching upper and lower space bounds for time-based and unnormalized sliding windows, demonstrating the generality and optimality of DS-FD across various sliding window models. This conclusively answers the open question regarding the optimal space bound for matrix sketching over sliding windows. We conduct extensive experiments with both synthetic and real-world datasets, validating our theoretical claims and thus confirming the correctness and effectiveness of our algorithm, both theoretically and empirically.

PVLDB Reference Format:
Hanyan Yin, Dongxie Wen, Jiajun Li, Zhewei Wei, Xiao Zhang, Zengfeng Huang, and Feifei Li. Optimal Matrix Sketching over Sliding Windows. PVLDB, 17(9): 2149 - 2161, 2024.
doi:10.14778/3665844.3665847

*Zhewei Wei is the corresponding author. The work was partially done at Gaoling School of Artificial Intelligence, Beijing Key Laboratory of Big Data Management and Analysis Methods, MOE Key Lab of Data Engineering and Knowledge Engineering, and Pazhou Laboratory (Huangpu), Guangzhou, Guangdong 510555, China.

Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.
doi:10.14778/3665844.3665847

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/yinhanyan/DS-FD.

1 INTRODUCTION

Many types of real-world streaming data, such as computer networking traffic, social media content, and sensor data, are continuously generated, often arriving in large volumes or at high speeds [17, 40]. Given the constraints in storage and computational resources, it becomes frequently impractical to store or compute aggregations and statistics for streaming data accurately. Algorithms for streaming data provide approximate solutions by summarizing, sketching, or synthesizing the data stream with sublinear space or time complexity relative to the input size [25, 37]. Among various streaming data algorithms, matrix sketching emerges as a general technique designed to process streaming data comprised of vectors or matrices [35]. A wide array of matrix sketching algorithms has been proposed, categorized into several approaches: sparsification [3, 15], sampling [3, 13, 27], random projection [28, 32], hashing [10, 34], and FrequentDirections (FD) [19, 21]. These algorithms typically present trade-offs between time-space complexity and accuracy. Notably, FrequentDirections [19, 21], a prominent deterministic algorithm, achieves a spectral bound on the relative covariance error, expressed as $\varepsilon = \|\boldsymbol{A}^\top\boldsymbol{A} - \boldsymbol{B}^\top\boldsymbol{B}\|_2/\|\boldsymbol{A}\|_F^2$, with an optimal space complexity of $O\left(\frac{d}{\varepsilon}\right)$. These attributes have led to its widespread application across various fields [8, 14, 16, 22].

In real-world scenarios, the interest often lies in the most recently arrived elements rather than outdated items within data streams, as highlighted by recent studies [11, 33, 38]. Datar et al. [12] consider the problem of maintaining aggregates and statistics from the most recent period of the data stream and refer to such a model as the *sliding window model*. This paper delves into the *continuous tracking matrix sketch over sliding windows*, a crucial technique for applications like sliding window PCA or real-time PCA [9, 33]. Such techniques play a vital role across various domains, including event detection [26], fault monitoring [1, 29], differential privacy [31] and online learning [18], highlighting the significance of optimizing both the complexity and the quality of estimations provided by matrix sketching algorithms for sliding windows.

Over the years, extensive research has been dedicated to developing improved sketching algorithms to address the challenge of

Table 1: Given the dimension d of each row vector, the upper bound of relative covariance error ε, and the size of the sliding window N, this table presents an overview of space complexities for algorithms addressing matrix sketching over sliding windows. An asterisk (*) indicates that the space complexity is the expected value when it is a random variable. For each column, *sequence-based* denotes that each update occupies a timestamp, *time-based* denotes that each timestamp unit may contain zero or multiple updates, *normalized* denotes the norm of each row equals a constant, and *unnormalized* denotes the norm of each row $\|\boldsymbol{a}_i\|_2^2 \in [1, R]$.

sketch κ	Sequence-based		Time-based	
	normalized	unnormalized	normalized	unnormalized
Sampling(SWR) [7, 33]	$O\left(\frac{d}{\varepsilon^2}\log N\right)$ *	$O\left(\frac{d}{\varepsilon^2}\log NR\right)$ *	$O\left(\frac{d}{\varepsilon^2}\log N\right)$ *	$O\left(\frac{d}{\varepsilon^2}\log NR\right)$ *
LM-HASH [10]	$O\left(\frac{d^2}{\varepsilon^3}\right)$	$O\left(\frac{d^2}{\varepsilon^3}\log R\right)$	$O\left(\frac{d^2}{\varepsilon^3}\log \varepsilon N\right)$	$O\left(\frac{d^2}{\varepsilon^3}\log \varepsilon NR\right)$
DI-RP [6]	$O\left(\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\right)$	$O\left(\frac{Rd}{\varepsilon^2}\log\frac{R}{\varepsilon}\right)$	-	-
DI-HASH [10]	$O\left(\frac{d^2}{\varepsilon^2}\log\frac{1}{\varepsilon}\right)$	$O\left(\frac{Rd^2}{\varepsilon^2}\log\frac{R}{\varepsilon}\right)$	-	-
LM-FD [21, 33]	$O\left(\frac{d}{\varepsilon^2}\right)$	$O\left(\frac{d}{\varepsilon^2}\log R\right)$	$O\left(\frac{d}{\varepsilon^2}\log \varepsilon N\right)$	$O\left(\frac{d}{\varepsilon^2}\log \varepsilon NR\right)$
DI-FD [21, 33]	$O\left(\frac{d}{\varepsilon}\log\frac{1}{\varepsilon}\right)$	$O\left(\frac{Rd}{\varepsilon}\log\frac{R}{\varepsilon}\right)$	-	-
DS-FD (This paper)	$O\left(\frac{d}{\varepsilon}\right)$	$O\left(\frac{d}{\varepsilon}\log R\right)$	$O\left(\frac{d}{\varepsilon}\log \varepsilon N\right)$	$O\left(\frac{d}{\varepsilon}\log \varepsilon NR\right)$
Lower bound (This paper)	$\Omega\left(\frac{d}{\varepsilon}\right)$	$\Omega\left(\frac{d}{\varepsilon}\log R\right)$	$\Omega\left(\frac{d}{\varepsilon}\log \varepsilon N\right)$	$\Omega\left(\frac{d}{\varepsilon}\log \varepsilon NR\right)$

matrix sketching over sliding windows. For instance, Wei et al. [33] proposed the Sampling, LM-FD, and DI-FD algorithms. Sampling is a probabilistic algorithm, and LM-FD and DI-FD are deterministic algorithms that build upon FrequentDirections. In addition, streaming matrix sketching algorithms based on *random projection* [6] and *hashing* [10], such as LM-HASH, DI-RP, and DI-HASH, are also compatible with the sliding window model. Zhang et al. [38] explored the challenge of tracking matrix approximations over distributed sliding windows, proposing communication-efficient algorithms like priority sampling, ES sampling, and DA. Braverman et al. [7] present a randomized row sampling framework for a wide spectrum of linear algebra approximation problems and a unified framework for deterministic algorithms in the sliding window model based on the merge-and-reduce paradigm and the online coresets. Furthermore, Shi et al. [30] also present a method to extend FrequentDirections to the persistent summary model, another historical range query model similar to the sliding window. Table 1 compares the memory cost of various matrix sketching algorithms over sliding windows.

Motivations. Despite the significant efforts mentioned above, existing algorithms for matrix sketching over sliding windows remain sub-optimal in terms of space complexity. Table 1 illustrates that for the fundamental case where the window is sequence-based (i.e., each update occupies a timestamp) and each row is normalized (i.e., the norm of each row equals a constant), the current best space complexity for matrix sketching algorithms over sliding windows is $O\left(\frac{d}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ for DI-FD and $O\left(\frac{d}{\varepsilon^2}\right)$ for LM-FD [33]. Conversely, it has been demonstrated in [19] that the space lower bound for FrequentDirections in the full stream model is $\Omega\left(\frac{d}{\varepsilon}\right)$, thus establishing the optimality of FrequentDirections. Given that the sliding window model poses greater challenges than the full stream model, $\Omega\left(\frac{d}{\varepsilon}\right)$ also represents a space lower bound for the sliding window model. This observation leads to a compelling inquiry: **Is it possible to achieve the optimal $O\left(\frac{d}{\varepsilon}\right)$ space bound for the problem of matrix sketching over the sliding window model?**

To address the question, it is crucial to recognize that the existing methods fail to achieve optimal space complexity primarily because they merely integrate FrequentDirections with generic sliding window algorithmic frameworks. This approach lacks a profound examination and enhancement of the original FrequentDirections algorithm specifically tailored for sliding windows. For instance, LM-FD applies FrequentDirections within the Exponential Histogram (EH) framework [12], and DI-FD combines FrequentDirections with the Dyadic Interval (DI) framework [33]. While these well-known frameworks are adept at adapting various streaming algorithms, such as Misra-Gries and SpaceSaving, to the sliding window model, they inherently introduce a multiplicative increase in memory overhead [20]. To develop more space-efficient sliding window algorithms, it is often necessary to undertake modifications directly on the streaming algorithms. This paper takes such an approach with FrequentDirections, aiming to refine and optimize it specifically for the sliding window context.

1.1 Our Contributions

In this paper, we introduce DS-FD, a deterministic algorithm that achieves optimal space complexity for matrix sketching over sliding windows. Specifically, DS-FD reaches an optimal space complexity of $O\left(\frac{d}{\varepsilon}\right)$ for sequence-based sliding windows with normalized rows. When the norms of the rows are not normalized and falling within the range of $[1, R]$, the space complexity naturally expands to $O\left(\frac{d}{\varepsilon}\log R\right)$. For time-based sliding windows, where each timestamp may not correspond to a row update, the space complexities are adjusted to $O\left(\frac{d}{\varepsilon}\log \varepsilon N\right)$ (row-normalized) and $O\left(\frac{d}{\varepsilon}\log \varepsilon NR\right)$

(row-unnormalized), respectively. These space complexities are detailed in the second-to-last row of Table 1.

Furthermore, we establish the lower bound of space complexity for any deterministic matrix sketching algorithm over sliding windows. Surprisingly, the corresponding lower bounds for the four models are also $\Omega\left(\frac{d}{\varepsilon}\right)$, $\Omega\left(\frac{d}{\varepsilon}\log R\right)$, $\Omega\left(\frac{d}{\varepsilon}\log \varepsilon N\right)$, and $\Omega\left(\frac{d}{\varepsilon}\log \varepsilon NR\right)$, as detailed in the last row of Table 1. This signifies that we have achieved optimal space complexity across the four distinct models. The specific contributions of this paper are outlined below:

- **Novel Algorithm with Improved Complexity:** We theoretically show the superiority of our DS-FD algorithm over existing methods in matrix sketching over sliding windows. Specifically, in the second-last column of Table 1, we highlight the theoretical improvements of DS-FD compared to existing methods. The space complexity $O\left(\frac{d}{\varepsilon}\right)$ of DS-FD is lower than that of two leading algorithms, $O\left(\frac{d}{\varepsilon^2}\right)$ for LM-FD and $O\left(\frac{d}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ for DI-FD. More importantly, our DS-FD algorithm implements modifications on the FrequentDirections based on our deeper insight into its application in sliding windows, rather than merely incorporating FrequentDirections into a generic sliding window framework, as done by LM-FD and DI-FD. Our DS-FD offers a deterministic error bound and applies to both sequence-based and time-based windows with an amortized update time of $O(d\ell)$, which is on par with the fastest FD in the full stream setting.
- **Matching Lower Bounds:** We establish matching lower bounds for any deterministic matrix sketching algorithm over sliding windows. Our proof is inspired by techniques used in the space lower bound proofs of BasicCounting [12] and other streaming matrix sketching problems [19]. This validation confirms that our DS-FD algorithm is optimal in terms of space complexity.
- **Extensive Experiments:** We conduct comprehensive experimental studies to verify the superiority of DS-FD over other state-of-the-art algorithms, especially in terms of sketch memory usage. Our experimental results reveal that the trade-off between error bounds and space cost for the DS-FD algorithm is more favorable than existing algorithms on synthetic and real-world datasets. Furthermore, optimizing space cost becomes increasingly significant as the permissible upper bound of covariance relative error tightens. These experimental findings are in strong agreement with our theoretical analyses.

2 PRELIMINARIES

This section introduces widely-used problem definitions of matrix sketching over sliding windows and several fundamental concepts related to this topic.

2.1 Problem Definition

Problem 1 (Matrix Sketching over Sliding Window). *Suppose we have a data stream where each item is in the set $\mathbb{R}^d$. Given the error parameter ε and window size N, the goal is to maintain a matrix sketch κ such that, at the current time T, κ can return an approximation $\boldsymbol{B}_W$ for the matrix $\boldsymbol{A}_W = \boldsymbol{A}_{T-N,T} \in \mathbb{R}^{N\times d}$, stacked by the recent N items. The approximation quality is measured by the covariance error, such that:*

$$\textbf{cova-error}(\boldsymbol{A}_W, \boldsymbol{B}_W) = \|\boldsymbol{A}_W^\top \boldsymbol{A}_W - \boldsymbol{B}_W^\top \boldsymbol{B}_W\|_2 \le \varepsilon \|\boldsymbol{A}_W\|_F^2,$$

where N bounds the maximum size of window W.

Wei et al. [33] show that maintaining a deterministic sketch with space less than $o(Nd)$ is not feasible if the maximum norm of row vectors of $\boldsymbol{A}_W$ is unbounded, even when a large covariance error is permissible. Consequently, contemporary research concentrates on problem variants where the maximum possible squared norm of all rows is upper-bounded. We define the following four variants of the FrequentDirections over sliding window problem. The paper will sequentially introduce algorithms to tackle these four problems, starting with the simplest, the sequence-based normalized window model, to illustrate the concept of dump snapshots. This approach serves as a foundation for addressing the unnormalized or the time-based model. We begin by formulating the problem for the sequence-based normalized window model, which is the simplest to solve and analyze.

Problem 1.1 (Matrix Sketching over Sequence-based Normalized Sliding Window). *We assume the squared norms of the rows in the window take value 1, that is, $\|\boldsymbol{a}\|_2^2 = 1$ for all $\boldsymbol{a} \in W$. Therefore, the covariance error is,*

$$\textbf{cova-error}(\boldsymbol{A}_W, \boldsymbol{B}_W) = \|\boldsymbol{A}_W^\top \boldsymbol{A}_W - \boldsymbol{B}_W^\top \boldsymbol{B}_W\|_2 \le \varepsilon \|\boldsymbol{A}_W\|_F^2 = \varepsilon N.$$

Next, we eliminate the normalization restriction, resulting in the following variant of the problem definition.

Problem 1.2 (Matrix Sketching over Sequence-based Sliding Window). *We assume the squared norms of the rows in the window range from $[1, R]$, that is, $1 \le \|\boldsymbol{a}\|_2^2 \le R$ for all $\boldsymbol{a} \in W$.*

If we account for real-world time instead of sequential order, the problems of both the normalized and unnormalized versions can be stated as follows.

Problem 1.3 (Matrix Sketching over Time-based Normalized Sliding Window). *We assume the squared norms of the rows in the window are either 0 or 1, that is, $\|\boldsymbol{a}\|_2^2 = 0$ or $\|\boldsymbol{a}\|_2^2 = 1$ for all $\boldsymbol{a} \in W$.*

Problem 1.4 (Matrix Sketching over Time-based Sliding Window). *We assume the squared norms of the rows in the window range from $\{0\} \cup [1, R]$, that is, $\|\boldsymbol{a}\|_2^2 = 0$ or $1 \le \|\boldsymbol{a}\|_2^2 \le R$ for all $\boldsymbol{a} \in W$.*

2.2 FrequentDirections

FrequentDirections [19, 21] is a deterministic algorithm of matrix sketching in the row-update model. It processes one row vector $\boldsymbol{a}_i \in \mathbb{R}^{1\times d}$ at a time, accumulating a stream of row vectors to construct a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, where $d \ll n$. Let $\ell = 1/\varepsilon$, FrequentDirections takes amortized $O(d\ell)$ time per row and maintains a sketch matrix $\boldsymbol{B} \in \mathbb{R}^{\ell\times d}$, achieving an error bound of

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

To process each arriving row vector, we first check whether $\boldsymbol{B}$ has any zero-valued rows. If $\boldsymbol{B}$ contains a zero-valued row, we insert $\boldsymbol{a}_i$ into it. Otherwise, we perform a Singular Value Decomposition (SVD) $[\boldsymbol{U}, \Sigma, \boldsymbol{V}^\top] = \mathsf{svd}(\boldsymbol{B})$, rescale the "directions" in $\boldsymbol{B}$ with the ℓ-th largest singular value σ_ℓ, and "forget" the least significant

direction in the column space of $V^\top$. The updated Σ' and sketch matrix B' are computed as follows:

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

and $B' = \Sigma' V^\top$, where $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_{2\ell}^2$. The updated B' will have at least ℓ+1 nonzero rows, allowing the process to continuously accommodate the next arriving row vector and update the sketch matrix B in the same manner as described above.

FrequentDirections can also be adapted for the sliding window model. We briefly introduce two algorithms based on it, which are the main competitors to our novel algorithms:

- LM-FD applies Exponential Histogram [12] to FrequentDirections. It organizes the window into exponentially shrinking block sizes, with the most recent level having blocks of size ℓ and the oldest level being $\varepsilon\|A_W\|_F^2$. As shown in Table 1, it uses $O\left(\frac{d}{\varepsilon^2}\right)$ space for sequence-based normalized sliding windows.
- DI-FD adopts the Dyadic Interval [2] approach to FrequentDirections, maintaining $L = \log \frac{R}{\varepsilon}$ parallel levels. At the j-th level, the matrix over the sliding window is segmented into sizes of $\Theta\left(\frac{NR}{2^{L-i}}\right)$. DI-FD uses $O\left(\frac{d}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ space for sequence-based normalized sliding windows.

3 OUR METHOD

In this section, we present a fundamental method that addresses Problem 1.1, termed **Dump Snapshots (DS)**. This method draws inspiration from the λ-snapshot method employed for solving the ε-approximate frequent items problem over the sliding window [20] and its connection to matrix sketching over the sliding window [19]. The algorithms developed for solving both the unnormalized and time-based problems are also based on this approach.

Connection to Item Frequency Estimation. The matrix sketching problem over sliding windows has a significant relation to the *ε-approximation frequent items* problem over sliding windows. The definition of this latter problem is outlined in [20, 39]:

Problem 2 (ε-approximation Frequent Items over Sliding Window). *Consider a data stream where each item belongs to the set $[d]$. Given an error parameter ε and a window size N, the goal is to maintain a sketch κ such that that ensures, for any item $i \in [d]$, the error between its true frequency f_i and estimated frequency $\hat{f}_i$ over the most recent N items returned by κ is constrained by:*

$$\left|f_i - \hat{f}_i\right| \leq \varepsilon N.$$

For the Problem 2, each element arriving in the data stream $A_W = \{a_1, a_2, \ldots, a_n\}$ can be viewed as a one-hot indicator vector, denoted as $a_i \in \{e_1, ..., e_d\}$, where e_j represents the j-th standard basis vector. The precise frequency of item j can be expressed as $f_j = \|A_W e_j\|_2^2$, which corresponds to the j-th element on the diagonal of matrix $A_W^\top A_W$. Assuming an algorithm that offers an estimated frequency $\hat{f}_i$ for each item i, we designate the i-th row of matrix B as $\hat{f}_i^{1/2} \cdot e_i$. Consequently, the error bound for the Problem 2, $\max_i \left|f_i - \hat{f}_i\right| \leq \varepsilon N$, can be expressed as $\|A_W^\top A_W - B_W^\top B_W\|_2 \leq \varepsilon N$, aligning with Problem 1.1. Hence, the Problem 2 over sliding window can be regarded as a special case of matrix sketching problem outlined in Problem 1.1, where the incoming row vectors are mutually orthogonal.

The Misra-Gries (MG) summary is a well-known algorithm for addressing the *ε-approximation frequent items* problem [23]. It operates by maintaining $1/\varepsilon$ counters, each initially set to 0. Upon encountering a new item, the algorithm increments its corresponding counter if it already has one; otherwise, it assigns a counter to it, setting its value to 1. If, following this increment, all $1/\varepsilon$ counters possess a value greater than 0, the algorithm decrements each counter by one, ensuring that there will be at least one counter reset to 0. This method effectively keeps track of the most frequent items within a data stream.

Extension of MG Summary to Sliding Windows. The MG summary can be effectively adapted to the sliding window model by the λ-snapshot method introduced by Lee et al. [20]. It preserves the error-bound guarantee without additional space or time complexity. This method's core concept is to log the event "item i has appeared εN times" along with the current timestamp and expire these logs in time. Similar to the original MG summary, the λ-snapshot method demands $O(1/\varepsilon)$ time for both updates and queries, and requires $O(1/\varepsilon)$ space. Moreover, an optimization has been proposed that can further diminish the time complexity for updates and queries to $O(1)$, while keeping the space complexity unchanged [39].

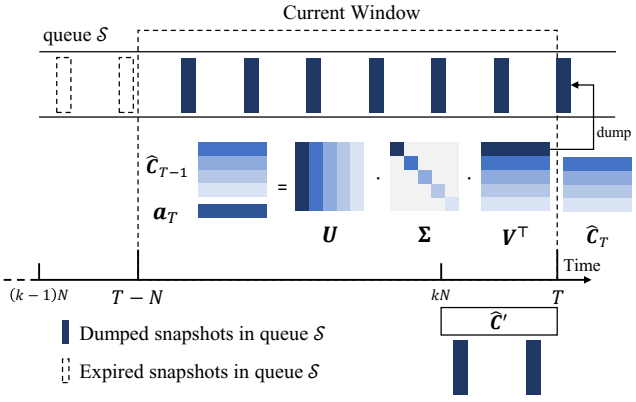


Figure 1: The data structures and update steps of DS-FD entail performing an SVD decomposition $U\Sigma V^\top = \text{svd}([\hat{C}_{T-1}, a_T])$ for each update. Following the decomposition, singular values and their corresponding right singular vectors are evaluated against the error bound εN. Those exceeding the bound are "dumped," i.e., removed from the current sketch and stored as snapshots in a queue $\mathcal{S}$, accompanied by the current timestamp.

Given the connection outlined above, where both the FD sketch and the λ-snapshot method derive from the principles of the MG summary, a compelling question emerges: Is it possible to extend the FD matrix sketch to the sliding window model while maintaining the same error-bound guarantee and the same space and time complexity as done by the λ-snapshot method? This question paves the way for the introduction of the DS-FD method.

Extension of FD Summary to Sliding Windows. The core concept of DS-FD is illustrated in Figure 1: for each FD update, after

performing an SVD $\left[\boldsymbol{U}, \Sigma, \boldsymbol{V}^\top\right] = \text{svd}(\left[\hat{C}_{T-1}, \boldsymbol{a}_T\right])$, we "dump" the singular values and their corresponding right singular vectors if the singular values exceed the error threshold εN. These vectors are stored as snapshots in a queue $\mathcal{S}$ with the timestamps. When a query is invoked, we combine the vectors from the snapshots within the current window's timeframe with the FD sketch to produce a matrix $\boldsymbol{B}_W$, which is then provided as the sketch.

Despite the simplicity of this approach, proving its correctness and achieving efficient implementation pose significant challenges. The main difficulties are: (1) The change in the orthogonal basis formed by the right singular vectors from the SVD of $\boldsymbol{A}_W$ before and after a single-step update, which results in past snapshots being non-orthogonal with current right singular vectors, complicating the error analysis. We prove the error bound for our algorithm as Theorem 3.1. (2) The challenge of accurately recording the timestamps of dumped snapshot vectors, which complicates the direct application of the FAST-FD algorithm. We also try to optimize it and propose the Fast-DS-FD.

The space requirement for DS-FD is $O(d/\varepsilon)$, aligning with the space lower bound of FD, thus confirming the optimality of our algorithm in terms of space usage. Additionally, we plan to extend our algorithms to address more general scenarios, including arriving vectors with norms $\|\boldsymbol{a}_i\|_2^2 \in [1, R]$, in Sections 4 and 5.

3.1 Algorithm Description

Algorithm 1: DS-FD: Initialize(d, ℓ, N, θ)

Input: d: Dimension of input vectors of Update
$\ell = \min\left(\lceil \frac{1}{\varepsilon} \rceil, d\right)$: Number of rows in FD sketch
N: Length of sliding window
θ: Dump threshold. For Problem 1.1, $\theta = \varepsilon N$

1 $\hat{C} \leftarrow \boldsymbol{0}_{\ell \times d}$
2 $\hat{C}' \leftarrow \boldsymbol{0}_{\ell \times d}$
3 Queue of snapshots $\mathcal{S} \leftarrow$ queue.Initialize()
4 Auxiliary queue of snapshots $\mathcal{S}' \leftarrow$ queue.Initialize()

Data Structures. Figure 1 and Algorithm 1 show the data structures of DS-FD. At any given moment, DS-FD maintains two FD sketches: a primary sketch $\hat{C}$ and an auxiliary sketch $\hat{C}'$. Each sketch is associated with its corresponding queue of snapshots, $\mathcal{S}$ and $\mathcal{S}'$, respectively. By setting $\ell = \min(d, \lceil 1/\varepsilon \rceil)$, the memory requirements for both the residual matrix $\hat{C}$ and the queue of snapshots $\mathcal{S}$ are $O(d/\varepsilon)$, leading to a total memory cost of $O(d/\varepsilon)$.
Update Algorithm. Algorithm 2 outlines the pseudocode for updating a DS-FD sketch, under the assumption that all arriving row vectors $\boldsymbol{a}_i$ are normalized, i.e., $\|\boldsymbol{a}_i\|_2^2 = 1$. The process starts by removing the oldest snapshots from the main queue until no snapshot in the queue is expired (lines 6-7). The input row vector then updates both the main and auxiliary FD sketches. If the squared norm of the top singular value multiplied by its corresponding right singular vector, $\|\hat{\boldsymbol{c}}_1\|_2^2$, in any FD sketch surpasses the error threshold (evaluated in lines 9 and 13), this component is recorded alongside the current timestamp as a snapshot at the end of the respective queue (lines 10 and 14), before its removal from the FD sketch (lines 11 and 15).

Algorithm 2: DS-FD: Update($\boldsymbol{a}_i$)

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

1 **if** $i \equiv 1 \mod N$ **then**
2 $\hat{C} \leftarrow \hat{C}'$
3 $\hat{C}' \leftarrow \boldsymbol{0}_{\ell \times d}$
4 $\mathcal{S} \leftarrow \mathcal{S}$
5 $\mathcal{S}' \leftarrow$ queue.Initialize()
6 **while** $\mathcal{S}[0].t + N \leq i$ **do** `// oldest snapshot expired`
7 $\mathcal{S}$.popleft()
8 $\hat{C} \leftarrow \text{FD}_\ell(\hat{C}, \boldsymbol{a}_i)$
9 **while** $\|\hat{\boldsymbol{c}}_1\|_2^2 \geq \theta$ **do**
10 $\mathcal{S}$ append snapshot $(\boldsymbol{v} = \hat{\boldsymbol{c}}_1, s = \mathcal{S}[-1].t + 1, t = i)$
11 Remove first row $\hat{\boldsymbol{c}}_1$ from $\hat{C}$ `// $\hat{\boldsymbol{c}}_2'$ becomes $\hat{\boldsymbol{c}}_1'$`
12 $\hat{C}' \leftarrow \text{FD}_\ell(\hat{C}', \boldsymbol{a}_i)$
13 **while** $\|\hat{\boldsymbol{c}}_1'\|_2^2 \geq \theta$ **do**
14 $\mathcal{S}'$ append snapshot $(\boldsymbol{v} = \hat{\boldsymbol{c}}_1', s = \mathcal{S}'[-1].t + 1, t = i)$
15 Remove first row $\hat{\boldsymbol{c}}_1'$ from $\hat{C}'$ `// $\hat{\boldsymbol{c}}_2'$ becomes $\hat{\boldsymbol{c}}_1'$`

Lines 1 to 5 describe the procedure of swapping the current main sketch and its queue with the auxiliary sketch and its queue, alongside initializing a new empty auxiliary sketch and queue of snapshots every N update step. This methodology, named as *restart every N steps*, contributes to maintaining the algorithm's error bound which is detailed in the proof of Theorem 3.1.

The predominant time complexity for Algorithm 2 arises from executing two FD updates, each necessitating an SVD on an $\ell \times d$ matrix. Hence, the total time complexity for each update step of DS-FD is $O(d\ell^2)$.
Optimized Update Algorithm. We introduce optimization strategies to enhance the update algorithm for the primary DS-FD. The efficiency of each update step in Algorithm 2 is dominated by the computation of $\text{FD}_\ell(\hat{C}, \boldsymbol{a}_i)$, which traditionally requires $O(d\ell^2)$ time. Ghashami et al. [19] have introduced the Fast-FD technique, effectively reducing the update time complexity of FD from $O(d\ell^2)$ to an amortized $O(d\ell)$. This optimization is achieved by performing a merge operation between the matrix formed by the arrival of ℓ vectors and the FD sketch every time ℓ vectors accumulate, updating the FD sketch with the merged result.

Expanding upon this principle, we propose the Fast-DS-FD algorithm, which decreases the time complexity for a single update operation in DS-FD from $O(d\ell^2)$ to an amortized $O(d\ell + \ell^3)$, without incurring additional spatial complexity. Particularly for scenarios where $d = \Omega(\ell^2)$, the amortized complexity is effectively $O(d\ell)$.

Algorithm 3 details the optimized update operation for Fast-DS-FD, adopting the Fast-FD strategy by postponing the SVD operation until ℓ rows are prepared for merging. This deferred condition is met once every ℓ iterations, allocating the majority of the computational effort to the SVD (line 6) and the computation of the new matrix $\boldsymbol{K} = \hat{C}\hat{C}^\top$ (line 10). Each of these steps requires $O(d\ell^2)$ time, thereby resulting in an amortized time cost of $O(d\ell)$. This approach significantly enhances the update mechanism within the sliding window framework, ensuring the algorithm remains efficient and effective.

Algorithm 3: Fast-DS-FD: Optimized UPDATE algorithm on Algorithm 2.

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

1 **while** $\mathcal{S}[0].t + N \le i$ **do** // oldest snapshot expired
2 $\quad \mathcal{S}$.POPLEFT()
3 $\boldsymbol{D} \leftarrow \begin{bmatrix} \hat{\boldsymbol{C}}^\top & \boldsymbol{a}_i^\top \end{bmatrix}^\top$
4 $\hat{\sigma_1} \leftarrow \sqrt{\hat{\sigma_1}^2 + \|\boldsymbol{a}_i\|_2^2}$
5 **if** *#(rows of* $\hat{D}$*)* $\ge 2\ell$ **then**
6 $\quad \hat{\boldsymbol{C}} \leftarrow \text{FastFD}_\ell(\boldsymbol{D})$
7 $\quad$ **while** $\|\hat{\boldsymbol{c}}_1\|_2^2 \ge \theta$ **do**
8 $\quad\quad \mathcal{S}$ append snapshot $(\boldsymbol{v} = \hat{\boldsymbol{c}}_1, s = \mathcal{S}[-1].t + 1, t = i)$
9 $\quad\quad$ Remove first row $\hat{\boldsymbol{c}}_1$ from $\hat{\boldsymbol{C}}$. // $\hat{\boldsymbol{c}}_2'$ becomes $\hat{\boldsymbol{c}}_1'$
10 $\quad \boldsymbol{K} = \hat{\boldsymbol{C}}\hat{\boldsymbol{C}}^\top$
11 $\quad \hat{\sigma_1} \leftarrow \|\hat{\boldsymbol{c}}_1\|_2$
12 **else**
13 $\quad \boldsymbol{K} \leftarrow \begin{bmatrix} \boldsymbol{K} & \boldsymbol{a}_i\hat{\boldsymbol{C}}^\top \\ \hat{\boldsymbol{C}}\boldsymbol{a}_i^\top & \boldsymbol{a}_i\boldsymbol{a}_i^\top \end{bmatrix}$
14 $\quad$ **if** $\hat{\sigma_1}^2 \ge \theta$ **then**
15 $\quad\quad [\boldsymbol{U}, \Sigma^2, \boldsymbol{U}^\top] \leftarrow \text{svd}(\boldsymbol{K})$
16 $\quad\quad$ **for** $j \in [1, \ell]$ **do**
17 $\quad\quad\quad$ **if** $\sigma_j^2 \ge \theta$ **then** // largest singular value
18 $\quad\quad\quad\quad \boldsymbol{v}_j^\top \leftarrow \frac{1}{\sigma_j}\boldsymbol{u}_j^\top \boldsymbol{D}$
19 $\quad\quad\quad\quad \mathcal{S}$ append snapshot $(\boldsymbol{v} = \boldsymbol{v}_j, s = \mathcal{S}[-1].t + 1, t = i)$
20 $\quad\quad\quad\quad \boldsymbol{D} \leftarrow \boldsymbol{D} - \boldsymbol{D}\boldsymbol{v}_j\boldsymbol{v}_j^\top$
21 $\quad\quad\quad\quad \boldsymbol{K} \leftarrow \boldsymbol{K} - (\boldsymbol{D}\boldsymbol{v}_j)(\boldsymbol{D}\boldsymbol{v}_j)^\top$
22 $\quad\quad \hat{\sigma_1} \leftarrow \sigma_1$
23 $\quad \hat{\boldsymbol{C}} \leftarrow \boldsymbol{D}$

In the general scenario where the "if" condition remains untriggered, it is essential to determine whether the maximum singular value σ_1 of the FD sketch $\boldsymbol{D} = \begin{bmatrix} \hat{\boldsymbol{C}}^\top & \boldsymbol{a}_i^\top \end{bmatrix}^\top$, after incorporating the newly arrived row vector $\boldsymbol{a}_i$, exceeds the threshold θ (line 17). If this criterion is met, the corresponding right singular vector $\boldsymbol{v}_1$ along with the current timestamp i is archived as a snapshot (line 19). This step also involves reducing the influence of this right singular vector $\boldsymbol{v}_1$ from both the FD sketch $\boldsymbol{D}$ and the covariance matrix $\boldsymbol{K}$, incurring a time cost of $O(d\ell + \ell^2)$ (lines 20, 21).

Rather than performing an SVD directly on the FD sketch $\boldsymbol{D}$ to compute $\boldsymbol{D} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$ within $O(d\ell^2)$, an incremental update to $\boldsymbol{K} = \boldsymbol{D}\boldsymbol{D}^\top$ is executed rank-1-wise in $O(d\ell)$ time (line 13). Following this, conducting SVD on $\boldsymbol{K}$ to obtain $\boldsymbol{K} = \boldsymbol{U}\Sigma^2\boldsymbol{U}^\top$ can be achieved in $O(\ell^3)$ (line 15). Then multiply the maximum singular value's corresponding left singular vector $\boldsymbol{u}_1$ with $\boldsymbol{D}$ to extract the right singular vector $\boldsymbol{v}_1^\top = \frac{1}{\sigma_1}\boldsymbol{u}_1^\top\boldsymbol{D}$ is completed in $O(d\ell)$ time (line 18).

Summarizing the aforementioned time expenditures, the amortized time complexity for a single update step in Algorithm 3 is established as $O(d\ell + \ell^3)$. Lemma 1 supports that the action taken in line 20—specifically, removing the j-th row of $\Sigma\boldsymbol{V}^\top$—parallels the procedure described in line 9.

LEMMA 1. *If* $\boldsymbol{D} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$ *and* $\boldsymbol{D}' = \boldsymbol{D} - \boldsymbol{D}\boldsymbol{v}_j\boldsymbol{v}_j^\top$*, where* $\boldsymbol{v}_j$ *is one of row vector of* $\boldsymbol{V}^\top$*. Then* $\boldsymbol{D} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top(\boldsymbol{I} - \boldsymbol{v}_j\boldsymbol{v}_j^\top)$*, which is same as remove the* j*-th row of* $\Sigma\boldsymbol{V}^\top$*.*

Given that the covariance matrix $\boldsymbol{K}$ might have m singular values surpassing the threshold θ, arranged as $\sigma_1^2 \ge \sigma_2^2 \ge \cdots \ge \sigma_m^2 \ge \theta$ increasingly, the "if" condition on line 17 within the loop could be validated m times, leading to a total operation count of $O(md\ell)$. For the normalized model, these operations can be averaged over $m\theta$ steps, yielding an amortized time complexity for the loop of $O(d\ell/\theta)$. With $\theta = N/\ell$ as specified in Problem 1.1, this amortization results in $O(d\ell^2/N)$. Assuming $N = \Omega(\ell)$, the amortized time complexity for a single update step in Algorithm 3 remains $O(d\ell + \ell^3)$, which is a reasonable assumption.

Setting $\ell = 1/\varepsilon$, the residual matrix $\hat{\boldsymbol{C}}$ of Fast-DS-FD can extend up to $2\ell \times d$, and the covariance matrix $\boldsymbol{K}$ can reach dimensions of $2\ell \times 2\ell$, leading to a space complexity of $O(\ell d + \ell^2)$. When $2\ell \le d$, the space complexity simplifies to $O(\ell d)$; for $2\ell > d$, maintaining the residual matrix $\hat{\boldsymbol{C}}$ up to $d \times d$ suffices, keeping the space complexity at $O(d^2 + \ell d) = O(\ell d)$. Thus, compared to DS-FD, Fast-DS-FD improves the update operation without incurring additional asymptotic space complexity.

Furthermore, potential optimizations could be realized. By maintaining an upper bound estimate $\hat{\sigma_1}$ on the maximum singular value σ_1 of the current FD sketch and updating this estimate to $\sqrt{\hat{\sigma_1}^2 + \|\boldsymbol{a}_i\|_2^2}$ with each new vector, an SVD is warranted only if this updated estimate exceeds the dump threshold θ (line 15), possibly avoiding the $O(\ell^3)$ SVD computation at this stage. Since only the largest singular value and its associated singular vector $\boldsymbol{u}_1$ of $\boldsymbol{K}$ are needed, iterative eigenvalue methods like Power Iteration could be used to reduce the time complexity of SVD further. Alternatively, randomized Krylov methods, such as Block Krylov, could offer a high-probability rank-1 approximation, requiring $O(\log(\min(1/\varepsilon, d)/\varepsilon'^{1/2}))$ matrix-vector multiplications, where ε' denotes the relative error of the rank-1 approximation. This adjustment might render Fast-DS-FD a probabilistic algorithm with an amortized update time complexity of $O(d\ell)$, marking a substantial efficiency enhancement [4, 24].

Query Algorithm. The query operation for DS-FD and Fast-DS-FD is elegantly simple, as delineated in Algorithm 4. It involves merging the FD sketch $\hat{\boldsymbol{C}}$ with matrix $\boldsymbol{B}$ stacked by the vectors of non-expiring snapshots. This process ensures that the query algorithm efficiently utilizes the preserved historical data within the sliding window, alongside the current sketch, to provide accurate matrix approximations.

Algorithm 4: DS-FD and Fast-DS-FD: QUERY()

1 **return** $\text{FD}_\ell(\boldsymbol{B}, \hat{\boldsymbol{C}})$, where $\boldsymbol{B}$ is stacked by $s_i.\boldsymbol{v}$ for all $s_i \in \mathcal{S}$

3.2 Analysis

We present the following theorem about the error guarantee, space usage, and update cost of DS-FD and Fast-DS-FD.

Theorem 3.1. *Assume that the data stream of vectors is* $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_T]$ *and* $\forall 1 \le i \le T, \|\boldsymbol{a}_i\|_2 = 1$. *Given the length* N *of the sliding window and the relative error* ε, *the DS-FD or Fast-DS-FD algorithm returns a sketch matrix* $\boldsymbol{B}_W$. *If we set* $\theta = \varepsilon N$ *and* $\ell = \min\left(\lceil \frac{1}{\varepsilon} \rceil, d\right)$, *then we have:*

$$\textbf{cova-err}(\boldsymbol{A}_W, \boldsymbol{B}_W) = \left\| \boldsymbol{A}_{T-N,T}^\top \boldsymbol{A}_{T-N,T} - \boldsymbol{B}_W^\top \boldsymbol{B}_W \right\|_2 \le 4\varepsilon N, \quad (1)$$

where $\boldsymbol{A}_W = \boldsymbol{A}_{T-N,T} = [\boldsymbol{a}_{T-N+1}, \boldsymbol{a}_{T-N+2}, \ldots, \boldsymbol{a}_T]^\top$. *The DS-FD or Fast-DS-FD algorithm uses* $O\left(\frac{d}{\varepsilon}\right)$ *space and process an update in* $O\left(d\ell^2\right)$ *time for DS-FD,* $O\left(d\ell + \ell^3\right)$ *time for Fast-DS-FD or* $O\left(d\ell\right)$ *time for probablistic Fast-DS-FD.*

The proof of Theorem 3.1 can be found in our technical report [36].

4 SEQUENCE-BASED MODEL

In this section, we delve into extending DS-FD to Seq-DS-FD to accommodate the general case where row vectors are not normalized, that is, $\|\boldsymbol{a}_i\|_2^2 \in [1, R]$, aiming to tackle Problem 1.2. Seq-DS-FD achieves the optimal space complexity of $O\left(\frac{d}{\varepsilon} \log R\right)$ for Problem 1.2, aligning with the problem's lower bound space complexity as established in Theorem 6.1.

4.1 High-Level Ideas

In the unnormalized scenario of sequence-based matrix sketching, the error bounds $\varepsilon\|\boldsymbol{A}_W\|_F^2$ fluctuate based on the distribution of the most recently arrived vectors $\boldsymbol{a} \in W$ over different periods. The minimum error bound is εN assuming that $\|\boldsymbol{a}\|_2^2 = 1$ for any $\boldsymbol{a} \in W$, whereas the maximum error bound reaches εNR, assuming $\|\boldsymbol{a}\|_2^2 = R$ for any $\boldsymbol{a} \in W$.

Drawing inspiration from the extension process applied in transitioning from the BasicCounting problem to the Sum problem [12], we regard the arrival of a row vector $\boldsymbol{a}_i$ with $\|\boldsymbol{a}_i\|_2^2 = v_i$ as analogous to the simultaneous arrival of v_i normalized vectors $\boldsymbol{a}_i/\sqrt{v_i}$. Following this approach, we apply the same update procedure as utilized in the normalized DS-FD. As a result, the outputs produced by the normalized DS-FD are confined within the window size of the most recent $\|\boldsymbol{A}_W\|_F^2 = \sum_{i=t_{\text{now}}-N+1}^{t_{\text{now}}} v_i$ normalized row vectors.

Given that the initial algorithm is limited to fixed-size sliding windows with a constant size of N, extending normalized DS-FD to general DS-FD effectively broadens the algorithm's capability to maintain and provide answers for matrix covariance estimation over sliding windows of variable sizes. Inspired by the extended λ-snapshot scheme for the FrequentItems problem across variable-size sliding windows [20], we introduce the Seq-DS-FD algorithm to address Problem 1.2. Illustrated in Figure 2, our approach involves maintaining $L = \lceil \log R \rceil$ layers of DS-FD structures concurrently, each configured with distinct error bounds and dump thresholds $\theta = 2^j \varepsilon N$ for the j-th level, where $0 \le j \le L$. Accordingly, the error bounds and dump thresholds extend from $\varepsilon N, 2\varepsilon N, \ldots, \varepsilon NR$ in an ascending sequence. The lower levels, characterized by smaller θ values, perform dump operations more frequently.

When handling query requests for a sketch $\boldsymbol{B}_W$ corresponding to the current window, we select the layer that meets the error bound criterion based on $\varepsilon\|\boldsymbol{A}_W\|_F^2$ for the prevailing window. Notably, it is unnecessary to actively maintain $\|\boldsymbol{A}_W\|_F^2$ or its approximation. Instead, we opt for the DS-FD layer that offers the minimal error bound while ensuring that the retained dumped snapshots span the time range of the current window size N. This chosen layer then yields the appropriate sketch $\boldsymbol{B}_W$.

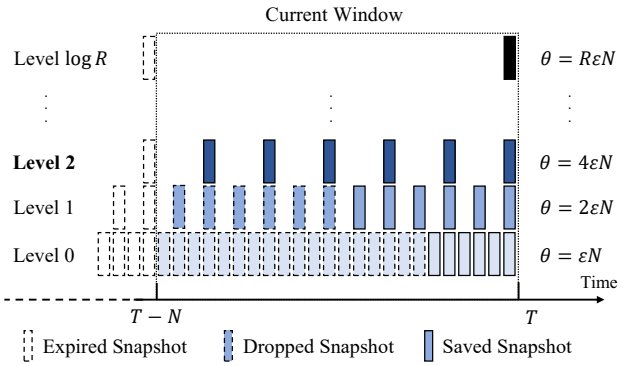


Figure 2: Sequence-based DS-FD. We maintain $L = \lceil \log R \rceil$ **layers of DS-FD structures in parallel, each with different error bounds and dump thresholds** $\theta = 2^j \varepsilon N$ **for the** j**-th level. In the visualization, depicted in dark blue, the norm of snapshots increases. For each level, we retain only the most recent** $O\left(\frac{1}{\varepsilon}\right)$ **snapshots saved in the queue and discard the older ones to limit the total memory usage to** $O\left(\frac{1}{\varepsilon} \log R\right)$.

4.2 Algorithm descriptions.

Algorithm 5: Seq-DS-FD: Initialize(d, ℓ, N, R, β)

Input: d: Dimension of input vectors of Update
$\ell = \min\left(\lceil \frac{1}{\varepsilon} \rceil, d\right)$: Number of rows in FD sketch
N: Length of sliding window
R: Upper bound of $\|\cdot\|_2^2$ for rows. For Problem 1.2, $R = \|A\|_{2,\infty}^2$
β: Additional coefficient of error, default as 1.0

1 $L \leftarrow \lceil \log_2 R \rceil$
2 $\mathcal{L} \leftarrow []$
3 **for** $j \in [0, L]$ **do**
4 $\quad \mathcal{L}$.append(Fast − DS − FD($d, \ell, N, \theta = 2^j \varepsilon N$))

Data Structures. As outlined in Algorithm 5, we determine the number of levels to be $L = \lceil \log_2 R \rceil$ (line 1). For each level i, we initialize a DS-FD structure with a dump threshold $\theta = 2^i \varepsilon N$ (lines 3-4). This setup specifies that the top row c_1 of the FD sketch is dumped as a snapshot if $\|c_1\|_2^2 \ge 2^i \varepsilon N$. Concurrently, to maintain memory efficiency, we cap the number of snapshots at each level to a maximum of $2(1 + 4/\beta)\frac{1}{\varepsilon}$, thereby constraining the memory requirement to $O\left(\frac{d}{\varepsilon} \log R\right)$.

Update Algorithm. Algorithm 6 details the procedure for updating a Seq-DS-FD sketch. Note that all incoming row vectors $\boldsymbol{a}_i$ have norms $\|\boldsymbol{a}_i\|_2^2$ within the range $[1, R]$. Initially, in line 3, we discard

Algorithm 6: Seq-DS-FD: UPDATE($\boldsymbol{a}_i$)

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

1 **for** $j \in [0, L]$ **do**
2 **while** $len(\mathcal{L}[j].\mathcal{S}) > 2(1 + \frac{4}{\beta})\frac{1}{\varepsilon}$ ***or*** $\mathcal{L}[j].\mathcal{S}[0].t + N \leq i$ **do**
3 $\mathcal{L}[j].\mathcal{S}$.POPLEFT()
4 **if** $\|\boldsymbol{a}_i\|_2^2 \geq 2^j \varepsilon N$ **then**
5 $\mathcal{L}[j].\mathcal{S}$ appends $(\boldsymbol{v} = \boldsymbol{a}_i, s = \mathcal{L}[j].\mathcal{S}[-1].t + 1, t = t_{\text{now}})$
6 $\mathcal{L}[j].\mathcal{S}'$ appends $(\boldsymbol{v} = \boldsymbol{a}_i, s = \mathcal{L}[j].\mathcal{S}'[-1].t + 1, t = t_{\text{now}})$
7 **else**
8 $\mathcal{L}[j]$.UPDATE($\boldsymbol{a}_i$)

the oldest snapshot from the queue until the head element of the queue remains within the current sliding window and ensure the snapshot count does not surpass $2(1 + 4/\beta)\frac{1}{\varepsilon}$. Subsequently, each level of DS-FD is updated with the input row vector $\boldsymbol{a}_i$. If the norm of the input row vector $\|\boldsymbol{a}_i\|_2^2$ is above the dump threshold θ (line 4), we directly save it as a snapshot and add it to the queue of the corresponding level (lines 5 and 6), effectively achieving zero error for this row. If not, the DS-FD is updated with $\boldsymbol{a}_i$ following the same procedures outlined in Algorithms 2 or 3 (line 8).

For every incoming row $\boldsymbol{a}_i$, Algorithm 6 processes the Fast-DS-FD sketch up to $\log R$ times across all $L = \log R$ layers, sum up to a total per-step update time of $O(d\ell \log R)$.

In Seq-DS-FD, it remains essential to concurrently manage dual sets of sketches for each level and execute a *restart every N step operation*, akin to the approach in DS-FD. Distinctively, for the primary DS-FD sketch at layer j within Seq-DS-FD, an alternating swap between the primary DS-FD sketch and the auxiliary DS-FD sketch occurs once the cumulative size of input vectors $\sum\|\boldsymbol{a}_i\|_F^2$ processed by the primary DS-FD sketch surpasses $2^{j+1}N$. This adaptation accounts for the perception of the arrival of a row vector $\boldsymbol{a}_i$ with $\|\boldsymbol{a}_i\|_2^2 = v_i$ as equivalent to the simultaneous arrival of $v_i/2^j$ rescaled vectors $\sqrt{\frac{2^j}{v_i}}\boldsymbol{a}_i$. For clarity, this mechanism is not explicitly depicted in the pseudocode of Algorithms 5 and 6.

Query Algorithm. Algorithm 7 details the procedure for generating a matrix sketch for the window $[t - N, t]$ utilizing a Seq-DS-FD sketch. Figure 2 illustrates the possible state of the data structure at a specific moment.

Given that the queue of each layer saves $O(\ell)$ snapshots, it is uncertain whether the snapshots preserved in the lower layers encompass the full window, as shown in levels 0 and 1 of Figure 2. Consequently, it is imperative to identify the lowest layer (offering minimal error) that contains snapshots spanning a window length of N (as exemplified by level 2 in Figure 2). A straightforward linear search could be conducted by verifying if the timestamp of the oldest snapshot in each layer's queue falls within the window interval $[t-N, t]$, which would entail a time complexity of $O(\log R)$. However, considering that the timestamp of the leading snapshot in each layer's queue increases monotonically, a binary search method could be employed, effectively reducing the time complexity to $O(\log \log R)$.

Algorithm 7: Seq-DS-FD: QUERY()

1 Find the $\min_j 1 \leq \mathcal{L}[j].\mathcal{S}[0].s \leq t_{\text{now}} - N + 1$
2 **return** $\text{FD}_\ell(\boldsymbol{B}, \mathcal{L}[j].\hat{C})$, where $\boldsymbol{B}$ is stacked by $s_i.\boldsymbol{v}$ for all $s_i \in \mathcal{L}[j].\mathcal{S}$

4.3 Analysis

We present the following theorem about the error guarantee, space usage, and update cost of Seq-DS-FD.

THEOREM 4.1. *Assume that the data stream of vector is* $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_T]$ *and* $\forall 1 \leq i \leq T, \|\boldsymbol{a}_i\|_2^2 \in [1, R]$. *Given the length N of the sliding window and the relative error* ε, *the Seq-DS-FD algorithm returns a sketch matrix* $\boldsymbol{B}_W$. *If we set* $\ell = \min\left(\lceil\frac{1}{\varepsilon}\rceil, d\right)$ *and* $\beta > 0$, *then we have:*

$$\textbf{cova-err}(\boldsymbol{A}_W, \boldsymbol{B}_W) = \left\|\boldsymbol{A}_W^\top \boldsymbol{A}_W - \boldsymbol{B}_W^\top \boldsymbol{B}_W\right\|_2 \leq \beta\varepsilon\|\boldsymbol{A}_W\|_F^2, \quad (2)$$

where $\boldsymbol{A}_W = \boldsymbol{A}_{T-N,T} = [\boldsymbol{a}_{T-N+1}, \boldsymbol{a}_{T-N+2}, \ldots, \boldsymbol{a}_T]^\top$. *Suppose the* β *is constant and u as the update time of each level; the Seq-DS-FD algorithm uses* $O\left(\frac{d}{\varepsilon}\log R\right)$ *space and processes an update in* $O(u \log R)$ *time.*

The proof of Theorem 4.1 can be found in our technical report [36].

5 TIME-BASED MODEL

In our previous discussions, we focused on *sequence-based data streams*, characterized by the regular arrival of data items, where the arrival time increments by one unit for each arrival. However, in many real-world applications of sliding window data streams, attention often shifts to sliding windows defined in real-time terms—such as maintaining a sketch of data items received over the past hour or day. This scenario is termed *time-based data streams*.

Time-based data streams differ from sequence-based streams primarily in two aspects: (1) The presence of *idle* periods, which denote intervals without any arriving items, or equivalently, when the incoming vector $\boldsymbol{a}_t$ is the zero vector. This characteristic introduces potential sparsity within the current time window, i.e., $\|\boldsymbol{A}_W\|_F^2 < N$ (assuming $\boldsymbol{a}_t \in \{0\} \cup [1, R]$). (2) The occurrence of *bursty* phenomena, characterized by abrupt increases in the rates of data item arrivals.

Adapting Seq-DS-FD to accommodate the time-based model is straightforward. Given that the error bound may be less than εN when $\|\boldsymbol{A}_W\|_F^2 < N$, we adjust the number of parallel DS-FD layers to $L = \lceil\log_2(\varepsilon NR)\rceil$, as opposed to $L = \lceil\log_2(R)\rceil$ employed in Seq-DS-FD. The dump thresholds are set to $\theta = 2^i$ $(1, 2, 4, \ldots, \varepsilon NR)$ for all layers $0 \leq i \leq L$, thereby determining the memory cost as $O\left(\frac{d}{\varepsilon}\log(\varepsilon NR)\right)$. The procedures for updates and queries remain consistent with those described in Algorithm 6 and Algorithm 7.

Here, NR may also represent the maximal potential value that $\|\boldsymbol{A}_W\|_F^2$ could achieve within a time-based sliding window. In instances where the arriving vectors are normalized, i.e., $R = 1$

and $\|\boldsymbol{a}_t\|_2^2 \in \{0,1\}$, the sketch's number of layers is modified to $L = \lceil \log(\varepsilon N) \rceil$, with the total size being $O\left(\frac{d}{\varepsilon}\log(\varepsilon N)\right)$.

Corollary 5.1. *Assume that the data stream of vectors is $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_T]$, and $\|\boldsymbol{a}_i\|_2^2 \in \{0\} \cup [1, R]$ for $\forall 1 \le i \le T$. Given the length N of the sliding window and the relative error ε, the Time-DS-FD algorithm returns a sketch matrix $\boldsymbol{B}_W$. If we set $\ell = \min\left(\lceil \frac{1}{\varepsilon} \rceil, d\right)$ and $\beta > 0$, then we have:*

$$\textbf{cova-error}(\boldsymbol{A}_W, \boldsymbol{B}_W) = \left\|\boldsymbol{A}_W^\top \boldsymbol{A}_W - \boldsymbol{B}_W^\top \boldsymbol{B}_W\right\|_2 \le \beta\varepsilon \|\boldsymbol{A}_W\|_F^2, \quad (3)$$

where $\boldsymbol{A}_W = \boldsymbol{A}_{T-N,T} = [\boldsymbol{a}_{T-N+1}, \boldsymbol{a}_{T-N+2}, \ldots, \boldsymbol{a}_T]^\top$. Suppose the β is constant and u as the update time of each level; the Time-DS-FD algorithm uses $O\left(\frac{d}{\varepsilon}\log \varepsilon NR\right)$ space and processes an update in $O(u \log \varepsilon NR)$ time.

6 SPACE LOWER BOUND

In this section, we delve into the lower bounds of space requirements for any deterministic algorithm designed to address the problem of matrix sketching over sliding windows. Our analysis aims to demonstrate that the space complexities of our proposed algorithms align with these lower bounds. This alignment confirms the optimality of our algorithms in terms of memory requirements, showcasing their efficiency in handling the constraints imposed by sliding window contexts.

Lemma 2. *Let $\boldsymbol{B}$ be a $\ell \times d$ matrix approximating a $n \times d$ matrix $\boldsymbol{A}$ such that $\|\boldsymbol{A}^\top \boldsymbol{A} - \boldsymbol{B}^\top \boldsymbol{B}\|_2 \le \|\boldsymbol{A} - \boldsymbol{A}_k\|_F^2/(\ell - k)$. For any algorithm with input as an $n \times d$ matrix A, the space complexity of representing $\boldsymbol{B}$ is $\Omega(d\ell)$ bits of space.*

Lemma 2, as established by Ghashami et al. [19], lays the groundwork for understanding the space efficiency of matrix sketching algorithms. Building upon this lemma, we aim to prove theorems concerning the space lower bounds for both sequence-based and time-based models of matrix sketching over sliding windows. These theorems ensure that our algorithms not only maintain precise sketching capabilities under the constraints of sliding window contexts but also adhere to the minimal possible space complexity.

Theorem 6.1 (Seq-based Lower Bound). *A deterministic algorithm that returns $\boldsymbol{B}_W$ such that*

$$\|\boldsymbol{A}_W^\top \boldsymbol{A}_W - \boldsymbol{B}_W^\top \boldsymbol{B}_W\|_2 \le \frac{\varepsilon}{3}\|\boldsymbol{A}_W\|_F^2,$$

where $\varepsilon = 1/\ell$, $\boldsymbol{A}_W \in \mathbb{R}^{N \times (d+1)}$, $N \ge \frac{1}{2\varepsilon}\log\frac{R}{\varepsilon}$ and $1 \le \|\boldsymbol{a}\|_2^2 \le R+1$ for all $\boldsymbol{a} \in \boldsymbol{A}_W$ must use $\Omega\left(\frac{d}{\varepsilon}\log R\right)$ bits space.

Proof. We partition a window of size N consisting of $(d+1)$-dimensional vectors into $\log R + 2$ blocks, as illustrated in Figure 3. The leftmost $\log R + 1$ blocks are labeled as $\log R, \ldots, 1, 0$ from left to right, as depicted in Figure 3. The construction of these blocks is as follows: (1) Choose $\log R + 1$ matrices of size $\frac{\ell}{4} \times d$ from a set of matrices $\mathcal{A}$, where $\mathcal{A}$ ensures that $\boldsymbol{A}_i^\top \boldsymbol{A}_i$ is an $\ell/4$ dimensional projection matrix and $\|\boldsymbol{A}_i^\top \boldsymbol{A}_i - \boldsymbol{A}_j^\top \boldsymbol{A}_j\| > 1/2$ for all $\boldsymbol{A}_i, \boldsymbol{A}_j \in \mathcal{A}$. Ghashami et al. [19] have demonstrated the existence of such a set $\mathcal{A}$ with cardinality $\Omega(2^{d\ell})$, making the total number of distinct arrangements $L = \binom{\Omega(2^{d\ell})}{\log R+1}$. Consequently, $\log L = \Omega(d\ell \log R)$. (2) For block i, multiply the chosen $\boldsymbol{A}_i \in \mathbb{R}^{\frac{\ell}{4} \times d}$ by a scalar of $\sqrt{\frac{2^i N}{\ell}}$, making the square of the Frobenius norm of block i, $\|\boldsymbol{A}_i\|_F^2$, equal to $2^i N/4$. (3) For block i where $i > \log \frac{\ell R}{N}$, increase the number of rows from $\ell/4$ to $\frac{\ell}{4} \cdot 2^{i - \log\frac{\ell R}{N}}$ to ensure that $1 \le \|\boldsymbol{a}\|_2^2 \le R$. The total number of rows is bounded by N, thus $\frac{N}{2} + \frac{\ell}{4}\log\frac{\ell R}{2N} \le N$, which implies $N \ge \frac{\ell}{2}\log \ell R$. (4) Set all the $(d+1)$-dimensional vectors in the window to be 1.

We assume the algorithm is presented with one of these L arrangements of length N, followed by a sequence of all one-hot vectors of length N with only the $(d+1)$-dimension set as 1. We denote $\boldsymbol{A}_W^i$ as the matrix over the sliding window of length N at the moment when $i+1, i+2, \ldots, \log R$ blocks have expired. Suppose our sliding window algorithm provides estimations $\boldsymbol{B}_W^i$ and $\boldsymbol{B}_W^{i-1}$ of $\boldsymbol{A}_W^i$ and $\boldsymbol{A}_W^{i-1}$, respectively, with a relative error of $\frac{1}{3\ell}$. This implies

$$\|{\boldsymbol{A}_W^i}^\top \boldsymbol{A}_W^i - {\boldsymbol{B}_W^i}^\top \boldsymbol{B}_W^i\|_2 \le \frac{1}{3\ell}\|\boldsymbol{A}_W^i\|_F^2 = \frac{1}{3\ell}\left(\frac{N}{4} \cdot 2^{i+1} + \frac{3N}{4}\right),$$

$$\|{\boldsymbol{A}_W^{i-1}}^\top \boldsymbol{A}_W^{i-1} - {\boldsymbol{B}_W^{i-1}}^\top \boldsymbol{B}_W^{i-1}\|_2 \le \frac{1}{3\ell}\|\boldsymbol{A}_W^{i-1}\|_F^2 = \frac{1}{3\ell}\left(\frac{N}{4} \cdot 2^{i} + \frac{3N}{4}\right).$$

Then we can answer the block i with $\boldsymbol{B}_i^\top \boldsymbol{B}_i = {\boldsymbol{B}_W^i}^\top \boldsymbol{B}_W^i - {\boldsymbol{B}_W^{i-1}}^\top \boldsymbol{B}_W^{i-1}$ as below,

$$\begin{aligned}
&\|\boldsymbol{A}_i^\top \boldsymbol{A}_i - \boldsymbol{B}_i^\top \boldsymbol{B}_i\|_2 \\
=&\|({\boldsymbol{A}_W^i}^\top \boldsymbol{A}_W^i - {\boldsymbol{A}_W^{i-1}}^\top \boldsymbol{A}_W^{i-1}) - ({\boldsymbol{B}_W^i}^\top \boldsymbol{B}_W^i - {\boldsymbol{B}_W^{i-1}}^\top \boldsymbol{B}_W^{i-1})\|_2 \\
\le&\|{\boldsymbol{A}_W^i}^\top \boldsymbol{A}_W^i - {\boldsymbol{B}_W^i}^\top \boldsymbol{B}_W^i\|_2 + \|{\boldsymbol{A}_W^{i-1}}^\top \boldsymbol{A}_W^{i-1} - {\boldsymbol{B}_W^{i-1}}^\top \boldsymbol{B}_W^{i-1}\|_2 \\
\le&\frac{1}{3\ell}\left(\frac{3}{4}N \cdot 2^i + \frac{3}{2}N\right) \le \frac{1}{\ell}\|\boldsymbol{A}_i\|_F^2.
\end{aligned}$$

The algorithm is capable of estimating the blocks for levels $0 \le i \le \log_2 R$. According to Lemma 2, each estimation of the blocks' levels by the algorithm necessitates $\Omega(d\ell)$ bits of space. Thus, we derive that a fundamental lower bound for the space complexity of any deterministic algorithm addressing the problem of matrix sketching over sliding windows is $\Omega\left(\frac{d}{\varepsilon}\log R\right)$. □

We also derive the space lower bound in the time-based model:

Theorem 6.2 (Time-based Lower Bound). *Any deterministic algorithm that returns $\boldsymbol{B}_W$ such that*

$$\|\boldsymbol{A}_W^\top \boldsymbol{A}_W - \boldsymbol{B}_W^\top \boldsymbol{B}_W\|_2 \le \frac{\varepsilon}{3}\|\boldsymbol{A}_W\|_F^2,$$

where $\boldsymbol{A}_W \in \mathbb{R}^{N \times d}$, $N \ge \frac{1}{2\varepsilon}\log\frac{R}{2}$ and $\|\boldsymbol{a}\|_2^2 \in 0 \cup [1, R]$ for all $\boldsymbol{a} \in \boldsymbol{A}_W$, one of the space lower bound is $\Omega\left(\frac{d}{\varepsilon}\log \varepsilon NR\right)$.

The proof of Theorem 6.2 is similar to that of Theorem 6.1 and can be found in our technical report [36].

Gathering Theorems 3.1, 4.1, 6.1, and 6.2, Corollary 5.1, Lemma 2, we synthesize the space complexity results of the optimal algorithms we've proposed against the proven space lower bounds for any deterministic algorithm under four distinct models in the last two rows of Table 1.

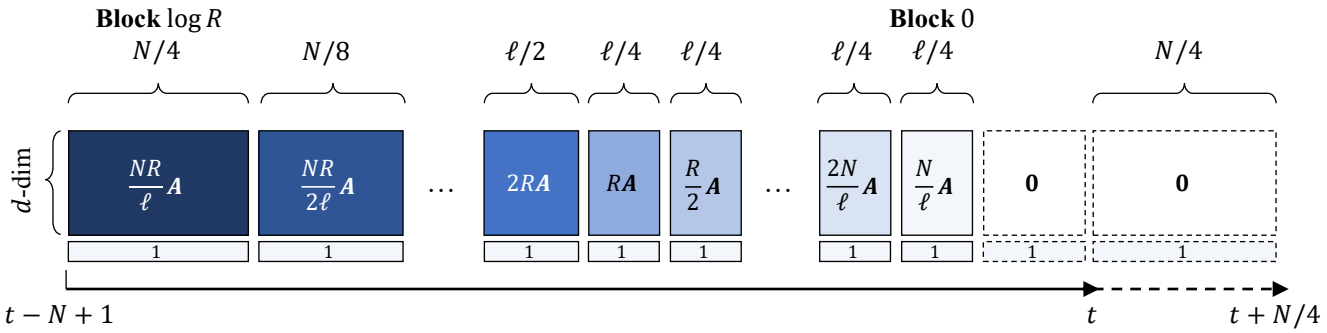


Figure 3: A constructive hard instance to establish a space lower bound for the sequence-based model. We initiate the sliding window's state by partitioning it into $\log R + 1$ blocks, each exponentially decreasing in size, and proceed to append one-hot vectors to the window over time. As each block expires, the algorithm is required to expend $\Omega(d\ell)$ bits to accurately estimate the expired block, according to Lemma 2. Consequently, by considering the number of blocks $\log R + 1$, we derive the lower bound $\Omega(d\ell \log R)$. The rigorous proof is provided in the text for Theorem 6.1.

7 EXPERIMENT

7.1 Experiment Setup

Datasets. We conduct our experiments on both sequence-based and time-based models, utilizing a combination of synthetic and real-world datasets. For the sequence-based model, our experiments encompass one synthetic dataset and two real-world datasets. The characteristics and sources of these datasets are detailed below and summarized in Table 2:

- **SYNTHETIC:** This dataset is a Random Noisy matrix commonly used to evaluate matrix sketching algorithms, generated by the formula $\boldsymbol{A} = \boldsymbol{SDU} + \boldsymbol{N}/\zeta$. Here, $\boldsymbol{S}$ is a $n \times d$ 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 $\boldsymbol{UU}^\top = \boldsymbol{I}_d$. The matrix $\boldsymbol{N}$ adds Gaussian noise, with $\boldsymbol{N}_{i,j}$ drawn from $\mathcal{N}(0,1)$. We set $\zeta = 10$ to ensure the signal $\boldsymbol{SDU}$ is recoverable. The window size is set to $N = 100,000$ for the SYNTHETIC dataset.
- **BIBD:**[1] This dataset is the incidence matrix of a Balanced Incomplete Block Design by Mark Giesbrecht from the University of Waterloo. It consists of 231 columns, 319,770 rows, and 8,953,560 non-zero entries, with each entry being an integer (0 or 1) indicating the presence or absence of an edge. The window size is set to $N = 10,000$ for the BIBD dataset.
- **PAMAP2 Physical Activity Monitoring:**[2] This dataset contains data from 18 different physical activities performed by 9 subjects wearing inertial measurement units and a heart rate monitor. For our experiments, we use data from subject 3, which includes 252,832 rows and 52 columns (timestamps and activity IDs removed, all missing entries set as 1). The window size is set to $N = 10,000$ for the PAMAP2 dataset.

We also evaluate the algorithms over the time-based model on two real-world datasets:

[1]University of Florida Sparse Matrix Collection
[2]UCI Machine Learning Repository

Table 2: Datasets for the sequence-based window.

Data Set	Total Rows n	d	N	Ratio R
SYNTHETIC	500,000	300	100,000	14.75
BIBD	319,770	231	10,000	1
PAMAP2	252,832	52	10,000	1,403

- **RAIL**[3] dataset is the crew scheduling matrix for the Italian railways, where the entry at row i and column j denotes the integer cost for assigning crew i to cover trip j. For our experiments, we selected a $200,000 \times 500$ submatrix. Synthetic timestamps for RAIL were generated following the *Poisson Arrival Process* with $\lambda = 0.5$, setting the window size to 50,000, which results in approximately 100,000 rows on average per window.
- **YearPredictionMSD (YEAR)**[4] is a subset from the "Million Songs Dataset" [5] that includes the prediction of the release year of songs based on their audio features. It comprises over 500,000 rows and $d = 90$ columns. For our analysis, a subset with $N = 200,000$ rows was utilized. This matrix exhibits a high rank. Synthetic timestamps for YEAR were similarly generated following the *Poisson Arrival Process* with $\lambda = 0.5$.

Table 3: Datasets for the time-based window.

Data Set	Total Rows n	d	Δ	N_W	Ratio R
RAIL	200,000	500	50,000	$\approx$ 100,000	12
YEAR	200,000	90	50,000	$\approx$ 100,000	1,321

Algorithms and Parameters. We compare DS-FD algorithm with three leading baseline competitors: Sampling algorithms, LM-FD, and DI-FD [33]. The evaluations of Sampling algorithms included SWR (row sampling with replacement) and SWOR (row sampling without replacement), where, for both strategies, sampling $\ell = O\left(d/\varepsilon^2\right)$ rows is required to attain an ε-covariance error.

[3]University of Florida Sparse Matrix Collection
[4]UCI Machine Learning Repository

The LM-FD algorithm is tested for sequence-based and time-based window models. For LM-FD, the space and error metrics were adjusted by a parameter $b = 1/\varepsilon$, the number of blocks per level, and a parameter $\ell = \min(1/\varepsilon, d)$, the size of the approximation matrix $\boldsymbol{B}$ (i.e., the number of rows in $\boldsymbol{B}$), to achieve an 8ε relative covariance error. The evaluation of DI-FD is only conducted to the sequence-based sliding window model, with the space and error parameters determined by $L = \log(R/\varepsilon)$, the maximal number of levels within the dyadic interval framework.

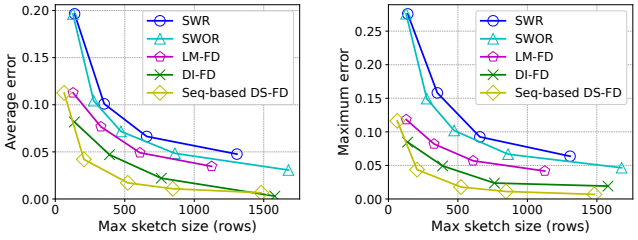
Furthermore, both Seq-DS-FD and Time-DS-FD are tested across sequence-based and time-based windows. Similarly to DI-FD, it is necessary to estimate the maximal value R of the row vector norm to set the number of layers as $L = \log R$ (for sequence-based) or $L = \log(\varepsilon NR)$ (for time-based). The parameters β and the size of FD sketches ℓ are adjustable, balancing space complexity and the covariance relative error boundary.

Metrics. In our experimental study, we adjust the parameters for each algorithm to illustrate the trade-offs between the *maximum sketch size* and the observe *maximum* and *average error*.

- **Sketch Size:** This metric denotes the space the matrix sketching algorithm occupies within the current window at a specific time. Considering that the primary part of the space cost comes from the row vectors of dimension d, we use the maximum number of rows to describe the space overhead for matrix sketching algorithms across different datasets.
- **Maximum and Average Relative Errors:** These metrics are employed to assess the quality of matrix estimates for various sketch algorithms. The relative error is defined as $\|\boldsymbol{A}_W^\top \boldsymbol{A}_W - \boldsymbol{B}_W^\top \boldsymbol{B}_W\|_2 / \|\boldsymbol{A}_W\|_F^2$, where $\boldsymbol{A}_W$ represents the accurate matrix within the current sliding window, and $\boldsymbol{B}_W$ is the estimated matrix produced by the sketching algorithm.

Hardware. For probabilistic algorithms such as random sampling, we employ the same random seed to guarantee the reproducibility of our experiments. All algorithms are implemented in Python 3.12.0. Experiments are conducted on a single idle core of an Intel® Xeon® CPU E7-4809 v4, clocked at 2.10 GHz.

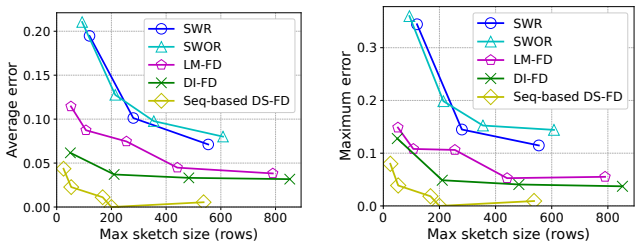
7.2 Experiment Results



(a) Average err vs. sketch size **(b) Maximum err vs. sketch size**

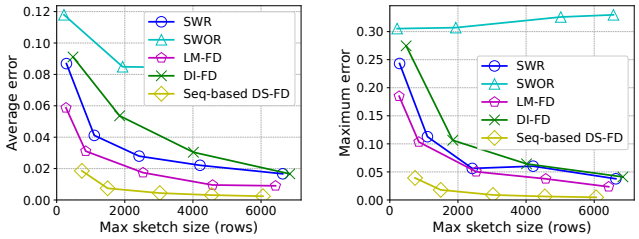
Figure 4: Error vs. sketch size on SYNTHETIC dataset.

Errors vs. Memory Cost. We begin our evaluation by comparing the empirical relative covariance error and memory cost across all algorithms. For each method, we adjusted a series of parameters to modulate the theoretical upper bound of the covariance error.



(a) Average err vs. sketch size **(b) Maximum err vs. sketch size**

Figure 5: Error vs. sketch size on BIBD dataset.



(a) Average err vs. sketch size **(b) Maximum err vs. sketch size**

Figure 6: Error vs. sketch size on PAMAP2 dataset.

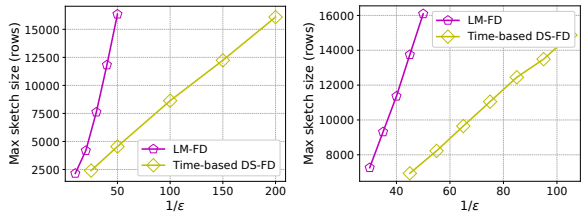
We report the maximum sketch sizes, along with the average and maximum empirical relative covariance error across all queries. The evaluation was conducted on the three datasets for the sequence-based sliding window model. Figures 4, 5, and 6 illustrate the trade-offs between maximum sketch size and average error, and between maximum sketch size and maximum error, respectively. From these observations, we infer the following points:

(1) In the sequence-based scenario, the error-space trade-off of LM-FD, DI-FD, and DS-FD outperforms that of the sampling algorithms SWR and SWOR. Notably, DI-FD exhibits a performance decline in skewed data streaming (PAMAP2), as depicted in Figures 6a and 6b, aligning with observations in [33].

(2) The trade-off between error and space for DS-FD is consistently superior to other competitors across both synthetic and real-world datasets in the sequence-based model. DS-FD achieves the same covariance error with less memory overhead than other methods. Furthermore, no empirical error was observed to exceed the theoretical bound, i.e., $\|\boldsymbol{A}_W^\top \boldsymbol{A}_W - \boldsymbol{B}_W^\top \boldsymbol{B}_W\|_2 > \varepsilon \|\boldsymbol{A}_W\|_F^2$, affirming our theoretical analysis and underscoring the efficiency and correctness of our algorithm.

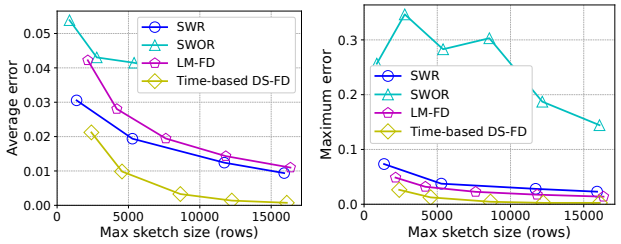
(3) The trade-off advantage of DS-FD becomes more pronounced as the ratio R (the ratio between the maximum and minimum squared norms in the dataset) decreases. For example, in Figures 5a and 5b, when row vectors are fully normalized in the BIBD dataset, DS-FD consumes significantly less memory to achieve a certain level of covariance error compared to other algorithms. Additionally, the average and maximum relative errors nearly approach 0 when the maximum sketch size exceeds 200 rows.

Subsequently, we evaluate four time-based algorithms on the two datasets tailored for time-based sliding windows. Figure 7 depicts



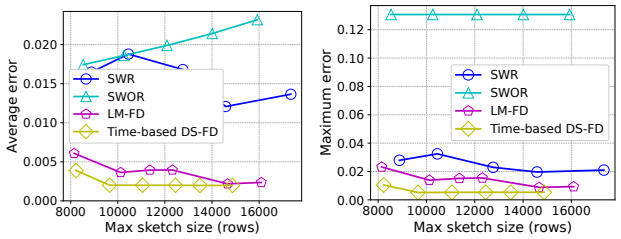
(a) RAIL (b) YEAR

Figure 7: Setups of parameter $1/\varepsilon$ vs. maximum sketch size of LM-FD and DS-FD.



(a) Average err vs. sketch size (b) Maximum err vs. sketch size

Figure 8: Error vs. sketch size on RAIL dataset.



(a) Average err vs. sketch size (b) Maximum err vs. sketch size

Figure 9: Error vs. sketch size on YEAR dataset.

the space overhead for LM-FD and Time-Based DS-FD with varying parameters ℓ on the RAIL and YEAR datasets. Additionally, Figures 8 and 9 illustrate the trade-offs between the maximum sketch size and the average error, and between the maximum sketch size and the maximum error, respectively. The experiments conducted on the time-based window model yield the following observations:

(1) As shown in Figure 7, the space cost of LM-FD escalates more rapidly than that of DS-FD as $1/\varepsilon$ increases. The actual space overheads for both LM-FD and Time-Based DS-FD align with their theoretical predictions of $O\left(\frac{d}{\varepsilon^2}\log \varepsilon NR\right)$ and $O\left(\frac{d}{\varepsilon}\log \varepsilon NR\right)$, respectively, corroborating the theoretical analyses.

(2) Figures 8 and 9 show that Time-Based DS-FD exhibits a superior space-error trade-off compared to other algorithms on both the RAIL and YEAR datasets. This indicates that our algorithm effectively adapts to the time-based sliding window model, maintaining its performance and efficiency.

Table 4: Update time and query time of all methods with a relative error bound of $\varepsilon = 1/100$ on the BIBD dataset.

Time(ms)	Update time	Query Time
SWR	65.722	157.500
SWOR	3.143	291.936
LM-FD	0.061	3599.310
DI-FD	2.428	59.904
DS-FD	1.053	27.655

Update and Query Time. We also record the average one-step update time and query time of all algorithms on BIBD dataset, as detailed in Table 4. Based on these observations, we draw the following conclusions:

(1) LM-FD requires the least amount of time for average updates, a finding that is in line with its update time complexity of $O(d \log \varepsilon NR)$ as reported by Wei et al. [33]. Conversely, the average query time for LM-FD is the highest among the evaluated methods. This increase in query time can be attributed to the time-consuming merging operation of all sketches in non-expiring blocks.

(2) Our DS-FD algorithm shows an acceptable average update time, while its average query time is the lowest among all the methods. This performance indicates that DS-FD effectively balances update and query times, making it an advantageous choice for matrix sketching over sliding windows.

8 CONCLUSION

In this paper, we delve into the challenge of matrix sketching over sliding windows, introducing a novel method, denoted as DS-FD. This method achieves space costs of $O\left(\frac{d}{\varepsilon}\log R\right)$ and $O\left(\frac{d}{\varepsilon}\log \varepsilon NR\right)$ for estimating covariance matrices in sequence-based and time-based sliding windows, respectively. Furthermore, we establish and validate a space lower bound for the covariance matrix estimation problem within sliding windows, demonstrating the space efficiency of our algorithm. Through extensive tests on large-scale synthetic and real-world datasets, we empirically validate the accuracy and efficiency of DS-FD, corroborating our theoretical analyses.

ACKNOWLEDGMENTS

This research was supported in part by National Science and Technology Major Project (2022ZD0114802), by National Natural Science Foundation of China (No. U2241212, No. 61932001, No. 62276066, No. 62376275), by Beijing Natural Science Foundation No. 4222028, by Beijing Outstanding Young Scientist Program (No.BJJWZYJH012019100020098), by Alibaba Group through Alibaba Innovative Research Program. 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, 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.

REFERENCES

[1] Mustapha Ammiche, Abdelmalek Kouadri, and Abderazak Bensmail. 2018. A modified moving window dynamic PCA with fuzzy logic filter and application to fault detection. *Chemometrics and intelligent laboratory systems* 177 (2018), 100–113.

[2] Arvind Arasu and Gurmeet Singh Manku. 2004. Approximate counts and quantiles over sliding windows. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 286–296.

[3] Sanjeev Arora, Elad Hazan, and Satyen Kale. 2006. A fast random sampling algorithm for sparsifying matrices. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 272–279.

[4] Ainesh Bakshi and Shyam Narayanan. 2023. Krylov Methods are (nearly) Optimal for Low-Rank Approximation. *arXiv preprint arXiv:2304.03191* (2023).

[5] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. 2011. The million song dataset. (2011).

[6] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. 2014. Near-optimal column-based matrix reconstruction. *SIAM J. Comput.* 43, 2 (2014), 687–717.

[7] Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P Woodruff, and Samson Zhou. 2020. Near optimal linear algebra in the online and sliding window models. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 517–528.

[8] Cheng Chen, Luo Luo, Weinan Zhang, Yong Yu, and Yijiang Lian. 2021. Efficient and robust high-dimensional linear contextual bandits. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 4259–4265.

[9] Ranak Roy Chowdhury, Muhammad Abdullah Adnan, and Rajesh K Gupta. 2020. Real-time principal component analysis. *ACM Transactions on Data Science* 1, 2 (2020), 1–36.

[10] Kenneth L Clarkson and David P Woodruff. 2017. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)* 63, 6 (2017), 1–45.

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

[12] 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.

[13] Amit Deshpande and Santosh Vempala. 2006. Adaptive sampling and fast low-rank matrix approximation. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 292–303.

[14] Charlie Dickens. 2020. Ridge Regression with Frequent Directions: Statistical and Optimization Perspectives. *arXiv preprint arXiv:2011.03607* (2020).

[15] Petros Drineas and Anastasios Zouzias. 2011. A note on element-wise matrix sparsification via a matrix-valued Bernstein inequality. *Inform. Process. Lett.* 111, 8 (2011), 385–389.

[16] Vladimir Feinberg, Xinyi Chen, Y Jennifer Sun, Rohan Anil, and Elad Hazan. 2024. Sketchy: Memory-efficient adaptive regularization with frequent directions. *Advances in Neural Information Processing Systems* 36 (2024).

[17] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. 2005. Mining data streams: a review. *ACM Sigmod Record* 34, 2 (2005), 18–26.

[18] Aurélien Garivier and Eric Moulines. 2011. On upper-confidence bound policies for switching bandit problems. In *International Conference on Algorithmic Learning Theory*. Springer, 174–188.

[19] 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.

[20] 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.

[21] 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.

[22] Luo Luo, Cheng Chen, Zhihua Zhang, Wu-Jun Li, and Tong Zhang. 2019. Robust frequent directions with application in online learning. *The Journal of Machine Learning Research* 20, 1 (2019), 1697–1737.

[23] Jayadev Misra and David Gries. 1982. Finding repeated elements. *Science of computer programming* 2, 2 (1982), 143–152.

[24] 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).

[25] Shanmugavelayutham Muthukrishnan et al. 2005. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science* 1, 2 (2005), 117–236.

[26] Mark Rafferty, Xueqin Liu, David M Laverty, and Sean McLoone. 2016. Real-time multiple event detection and classification using moving window PCA. *IEEE Transactions on Smart Grid* 7, 5 (2016), 2537–2548.

[27] Mark Rudelson and Roman Vershynin. 2007. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM)* 54, 4 (2007), 21–es.

[28] Tamas Sarlos. 2006. Improved approximation algorithms for large matrices via random projections. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*. IEEE, 143–152.

[29] M Ziyan Sheriff, Majdi Mansouri, M Nazmul Karim, Hazem Nounou, and Mohamed Nounou. 2017. Fault detection using multiscale PCA-based moving window GLRT. *Journal of Process Control* 54 (2017), 47–64.

[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] Jalaj Upadhyay and Sarvagya Upadhyay. 2021. A framework for private matrix analysis in sliding window model. In *International Conference on Machine Learning*. PMLR, 10465–10475.

[32] Santosh S Vempala. 2005. *The random projection method*. Vol. 65. American Mathematical Soc.

[33] 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.

[34] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*. 1113–1120.

[35] David P Woodruff et al. 2014. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science* 10, 1–2 (2014), 1–157.

[36] Hanyan Yin, Dongxie Wen, Jiajun Li, Zhewei Wei, Xiao Zhang, Zengfeng Huang, and Feifei Li. 2024. Optimal Matrix Sketching over Sliding Windows. arXiv:2405.07792 [cs.DB]

[37] Tianjing Zeng, Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. 2022. Persistent Summaries. *ACM Transactions on Database Systems (TODS)* 47, 3 (2022), 1–42.

[38] 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.

[39] Linfeng Zhang and Yong Guan. 2008. Frequency estimation over sliding windows. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 1385–1387.

[40] Zhi-Hua Zhou. 2023. Stream efficient learning. *arXiv preprint arXiv:2305.02217* (2023).