

Robust Streaming Against Low-Memory Adversaries

Omri Ben-Eliezer ✉ 

Faculty of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel

Krzysztof Onak ✉ 

Faculty of Computing & Data Sciences, Boston University, MA, USA

Sandeep Silwal ✉ 

Department of Computer Sciences, University of Wisconsin-Madison, WI, USA

Abstract

Robust streaming, the study of streaming algorithms that provably work when the stream is generated by an adaptive adversary, has seen tremendous progress in recent years. However, fundamental barriers remain: the best known algorithm for turnstile F_p -estimation in the robust streaming setting is exponentially worse than in the oblivious setting, and closing this gap seems difficult. Arguably, one possible cause of this barrier is the adversarial model, which may be too strong: unlike the *space-bounded* streaming algorithm, the adversary can memorize the entire history of the interaction with the algorithm. Can we then close the exponential gap if we insist that the adversary itself is an *adaptive but low-memory* entity, roughly as powerful as (or even weaker than) the algorithm?

In this work we present the first set of models and results aimed towards this question. We design efficient robust streaming algorithms against adversaries that are fully adaptive but have no long-term memory (“memoryless”) or very little memory of the history of interaction. Roughly speaking, a memoryless adversary only sees, at any given round, the last output of the algorithm (and does not even know the current time) and can generate an unlimited number of independent coin tosses. A low-memory adversary is similar, but maintains an additional small buffer. While these adversaries may seem quite limited at first glance, we show that this adversarial model is strong enough to produce streams that have high flip number and density in the context of F_2 -estimation, which rules out most known robustification techniques. We then design a new simple approach, similar to the computation paths framework, to obtain efficient algorithms against memoryless and low-memory adversaries for a wide class of order-invariant problems. We conclude by posing various open questions proposing further exploration of the landscape of robust streaming against fully adaptive but computationally constrained adversaries.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Streaming models

Keywords and phrases robust streaming, adaptive robustness, bounded-space adversaries

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.16

Related Version *Full Version*: <https://arxiv.org/abs/2511.01769>

Funding *Omri Ben-Eliezer*: Supported by an Alon Scholarship and by a Taub Family Foundation “Leaders in Science & Technology” Fellowship.

1 Introduction

Adversarially robust streaming, i.e., the study of streaming algorithms that are robust against adaptive adversaries, has received a great deal of attention in the last few years (see, e.g., [15, 6, 5, 16, 31, 7, 4, 11, 12, 28, 10, 13, 30, 14, 24, 25, 2, 21, 27] and the many references within). In the streaming setting, the goal of the algorithm is to approximately compute some



© Omri Ben-Eliezer, Krzysztof Onak, and Sandeep Silwal;
licensed under Creative Commons License CC-BY 4.0

17th Innovations in Theoretical Computer Science Conference (ITCS 2026).

Editor: Shubhangi Saraf; Article No. 16; pp. 16:1–16:23



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

function over the stream – say, the F_p -moment¹ – using as little space as possible, ideally using memory that is polylogarithmic in the stream length m and the underlying universe size n , and polynomial in the inverse of the approximation parameter ϵ . In the standard (black box) adversarially robust streaming model, the stream is generated by an adversary who sees and memorizes all outputs of the algorithm along the stream. The adversary can choose each stream update in a way that depends on the *entire history* of the interaction between the adversary and the algorithm – that is, on all stream updates and algorithm outputs so far. This is in contrast to the standard oblivious model, in which the stream updates do not depend on the actions of the streaming algorithm (i.e., the stream is thought of as generated beforehand and is not adaptive).

Arguably the most interesting open question in adversarially robust streaming concerns the space complexity of turnstile moment estimation (F_p -estimation), in particular for $p = 0$ and $p = 2$ [17]. In the turnstile streaming setting, both insertions and deletions of elements are allowed. For simplicity, we shall consider the setting of unit updates, in which each stream update either inserts or deletes a single copy of one item chosen by the adversary (and thus changes the frequency of the item by $+1$ or -1 , respectively). Unfortunately, there are exponential gaps between the optimal space complexity against oblivious adversaries and the best-known streaming algorithms in the presence of adaptive adversaries. In the oblivious case, algorithms based on linear sketching attain space complexity that is logarithmic in m and n for F_p -estimation when $p \leq 2$. Meanwhile, in the adversarially robust regime, all existing solutions have space complexity polynomial in the length m of the stream – an exponentially worse dependence in m . For example in F_2 -estimation, the best known algorithms [4, 30] require space complexity $\tilde{O}(m^{0.4})$, which is exponentially worse than that of linear sketches in oblivious streams. Thus, closing this gap between oblivious and robust turnstile F_p -streaming is a central and likely challenging open problem [17].

Upon a closer inspection of the robust streaming setting, however, one can easily see that there is an inherent power imbalance between the algorithm and the adversary in this model. While the streaming algorithm is required to use a small amount of space (much smaller than m , the length of the stream), the adversary inherently has memory of size at least m , as it retains all previous stream updates and all previous outputs of the algorithm.

In this work we set out to explore streaming models that maintain a better balance between the respective computational power (in this case, the amount of memory) of the algorithm and that of the adversary. Our main high-level motivating question is:

Is it possible to obtain robust streaming algorithms for, say, F_2 -estimation against adversaries with memory size k , where the space complexity of the algorithm is polylogarithmic in m and n , and polynomial in $1/\epsilon$ and k ?

While we are not able to fully resolve the above question (and it remains a very intriguing open problem), we obtain the first positive results of this flavor.

Importantly, there are other beyond worst case models in the literature that constrain the adversary. Examples include models in which the adversary has a limited amount of adaptive interaction with the algorithm (i.e., it can modify the future input stream only a limited number of times) [28, 10]; or models in which the algorithm occasionally receives a small amount of advice from a powerful (and not memory-constrained) oracle [28]. However, none of these models capture the natural setting considered in this paper, where the adversary is fully adaptive – that is, it sees all outputs of the algorithm and can immediately edit the next input after each algorithm’s output – but constrained in terms of its internal memory size.

¹ Assuming that the elements in the stream come from a finite universe, such as $[n]$, the F_p -moment of the stream is $\sum_{i \in [n]} |f_i|^p$, where f_i is the frequency (possibly negative) of item i in the stream.

1.1 Our Model

Let us first quickly recall the standard adversarial model [6, 5]. At a high level, the streaming algorithm receives m updates of items (insertions and deletions) from a finite universe of size n . In general, n can be much bigger than m , but without loss of generality, one can assume that $n = O(\text{poly}(m))$, because one can randomly hash the stream elements into $O(\text{poly}(m))$ buckets, while ensuring no collisions for different items with high probability (with the caveat that one would also need space to store the hash function). Thus, for simplicity, we assume items are elements in the domain $[n]$. A key concept is the notion of a *frequency vector* of the underlying stream, which simply records the number of times every item has appeared (where a deletion is counted as a negative occurrence, see Definition 6 for details).

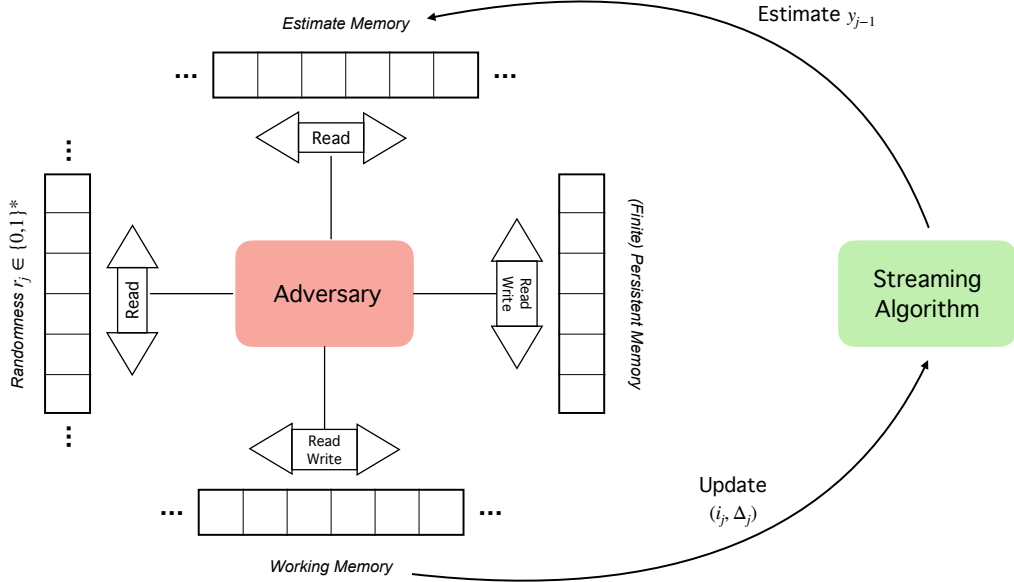
The goal of the streaming algorithm is to estimate functions of the underlying stream, where at any point in the stream, the function only depends on the current frequency vector. We consider the *tracking* variant in which the algorithm must output an estimate after every stream update. Perfect estimation is of course possible by exactly recording the frequency vector, but the goal of the entire field of streaming algorithms is to design algorithms that operate in space sublinear in the size of the input (in our case, ideally poly-logarithmic in n and m). This setting is quite general and captures a wide range of streaming problems, including F_p -estimation and graph streaming, among others [26, 23].

The bounded-memory adversarial model. In our adversarial setting, the stream updates are controlled by a possibly randomized adversary. Crucially, the adversary is memory-bounded and its actions only depend on the *most recent* estimate of the algorithm, with possibly a very small amount of longer-term memory. To model this, we define several types of memory for the adversary. See Figure 1 for a visual depiction and Section 3 for a formal definition of the model.

- **Estimate memory:** We assume the adversary has an *estimate memory* where the most recent estimate of the algorithm (and nothing else) is always written. (We do not bound the amount of this memory but for problems that we consider, a good approximation to a function's value can easily be provided in small memory, using the standard number representation.)
- **Working memory:** We equip the adversary with a *working memory*, which is also unbounded. This allows the adversary to compute an arbitrary function of the latest estimate in the estimate memory. Note that this memory would easily allow the adversary to write down and remember the entire history of the algorithm's outputs, thus reducing our setting to the general robust streaming one. Thus, in our setting, the working memory is wiped clean after the adversary computes the next update for the streaming algorithm.
- **Persistent memory:** The adversary may have a (limited amount of) *persistent memory* which *does* allow the adversary to store and update a bounded amount of information throughout the entire stream. If the persistent memory is limited to zero bits, then we refer to the adversary as *memoryless*.
- **Randomness:** Lastly, we also allow the adversary to use as much randomness as it desires, and assume it has access to random bits freshly drawn at every round. These freshly generated random bits are not retained for future rounds.² Thus, a memoryless

² One could also consider allowing the adversary to generate and memorize an unlimited amount of randomness in the beginning of the process, before any interaction with the adversary. However, as we discuss in Remark 12, this would not effectively change the power of the adversary.

algorithm samples an update from some distribution P_y that depends on the latest estimate y provided by the algorithm. For a low-memory algorithm, this distribution depends on the last output and on the current state of the persistent memory. As usual, a memoryless (or bounded-memory) adversary is deterministic if it does not use any random bits during its operation.



■ **Figure 1** A pictorial representation of our streaming model. Not shown: The estimate memory is overwritten by the estimate y_{j-1} of the streaming algorithm, and the working memory is wiped clean after each update (i_j, Δ_j) is computed. We also don't show the inner mechanisms of the streaming algorithm or the potential strategy the adversary is following. See Definition 11 for the formal presentation.

1.2 Our Results

Robust streaming against deterministic memoryless adversaries. The weakest fully-adaptive adversary whom one could imagine in our context is a deterministic memoryless adversary. Note that for this adversary, the random read-only memory and the persistent memory are empty, and the working memory starts each round empty as well. So the adversary makes the decision what update to send only based on the last output of the algorithm. Through the use of a standard rounding-based mechanism and sparsity arguments, we obtain the following.

► **Theorem 1** (Deterministic adversaries; informal version of Theorem 14 and Corollary 15). *There exists a deterministic streaming algorithm that, for a fixed function f of the underlying stream frequency vector with range $\{0\} \cup [1, \alpha]$, tracks the value of f up to a multiplicative factor of $1 + \epsilon$ against a deterministic memoryless adversary, using $O\left(\frac{\log(\alpha)}{\epsilon} \cdot \log(mn)\right)$ bits of space. If the adversary is deterministic and has k bits of persistent memory, the space bound is $O\left(2^k \cdot \frac{\log(\alpha)}{\epsilon} \cdot \log(mn)\right)$.*

We note that the above theorem poses almost no constraints on the function f – that is, essentially *any* reasonable function of the frequency vector (beyond just specific special cases, such as F_p -moments) can be efficiently and deterministically approximated against a deterministic adversary. These results are proved in Section 4.

Memoryless adversaries can generate high flip-number, high density streams. The next natural question is whether it is possible to design adversarially robust algorithms against *randomized* low-memory adversaries. Before proving the positive results on this front, we demonstrate that the class of streams generated by these adversaries is sufficiently interesting and non-trivial to resist a black-box application of existing robust streaming frameworks in the literature. Specifically, we prove that such an adversary can generate streams that have high *density*³ and large *flip number*,⁴ in the case where f is F_2 , the second moment of the frequency vector.

► **Theorem 2** (Low-memory stream with high density and large flip number; informal version of Theorem 16). *Fix any constant $0 < c < 1$. There exists a randomized memoryless adversary for F_2 -estimation that, conditioned on the algorithm always outputting a $(1+\epsilon)$ -approximation in the specific instance of the adversarial game, generates a stream of length m whose density is $\Omega(m^c)$, for all but the first m^c rounds of the stream, and whose flip number is $\Omega(m^{1-c})$.*

In addition, if we provide the adversary with a single bit of persistent memory, then the above statement holds, for any constant $0 < c < 1$, with flip number $\Omega(m^{1-0.5c})$.

Again the result here is rather general in a certain interesting sense. While the result applies only to the special case of $f = F_2$, it only concerns the adversary and does not assume anything specific about the behavior of the algorithm, aside from requiring that it provides good estimates throughout the stream. In particular, it also applies to algorithms that can memorize the entire stream.

We note that it is easy to construct a stream with flip number $\Omega(m)$, but density $O(1)$, using a deterministic memoryless adversary: simply take an adversary who when the algorithm outputs zero, inserts the element, say, 1; and when the algorithm outputs any value bigger than zero, the adversary deletes the same element. It is also easy to construct a stream with flip number of order $\Theta(\log m)$ and density $\Omega(m)$, this time with a randomized memoryless adversary (provided the underlying universe is large enough): simply insert a random element from the universe in each round. However, each of these examples in itself is not very interesting and can be easily handled by at least one of the existing generic black-box frameworks for robust streaming. But as it turns out, these two simple constructions can be interwoven together (by a randomized memoryless adversary) to produce a stream with both high (polynomial in m) flip number and high density.

The state of the art robust streaming algorithms for F_2 -estimation against *any* black-box adversary (not just bounded adversaries as in this paper) have space complexity $\tilde{\Theta}(m^{0.4})$ and are based on dense-sparse tradeoffs. The earlier and simpler of them, by Ben-Eliezer, Eden, and Onak [4], has space complexity that scales with the maximum between the sparsity and the square root of the flip number for the non-sparse part of the stream. The more recent one, due to Woodruff and Zhou [30], is more intricate, attains the same space complexity for F_2 -estimation (and improved complexity for F_p with $p \in (1, 2)$) but is better tailored for

³ The density of the stream (or more precisely, of the frequency vector of the stream) at any point during the process is the number of non-zero entries in the frequency vector at that time. See Definition 10.

⁴ The flip number is roughly the number of times the underlying value of f over the stream changes by a constant multiplicative factor, see Definition 9.

streams in which the dense part involves insertions and deletions on a small set of heavy hitters. Informally speaking, our construction toward the proof of Theorem 16 appears to pose a barrier for the algorithm of Ben-Eliezer et al. (and more generally any algorithm whose complexity depends as a black box on the sparsity of the stream and the flip number of the dense part), but the Woodruff-Zhou algorithm appears better tailored for processing such a stream efficiently since the dense part in our construction involves repeated insertions and deletions of the same single element. Nevertheless, the fact that memoryless adversaries can generate streams with both large flip number and large density provides a strong indication that such a model, albeit seemingly weak at first glance, is interesting and strong enough to fool many of the existing approaches in the literature. We leave open the question of whether the Woodruff-Zhou algorithm (or variants thereof) can solve F_2 -estimation against memoryless adversaries with space complexity polylogarithmic in m (see Section 7). Even if this is the case, our algorithms have the advantage of being significantly simpler.

Handling memory: bounded randomized adversaries. We now go beyond the deterministic case and turn our attention to randomized adversaries. Our main theorem is the following.

► **Theorem 3** (Randomized adversaries; informal version of Theorem 27). *Suppose there exists an order-independent oblivious streaming algorithm that uses $M(\epsilon)$ space to compute a $(1 + \epsilon)$ -approximation to a function f with range $\{0\} \cup [1, \alpha]$ with probability at least $9/10$.*

Then, there exists a corresponding robust algorithm with space $M(\epsilon/3) \cdot O\left(\frac{\log \alpha}{\epsilon} \cdot \log m\right)$ that tracks the value of f up to a factor of $1 + \epsilon$ against a memoryless randomized adversary, with probability at least $9/10$ over the entire stream. If the adversary has k bits of persistent memory, the space bound increases to $M(\epsilon/3) \cdot O\left(2^k \cdot \frac{\log \alpha}{\epsilon} \cdot \log m\right)$.

To prove the above theorem, we first introduce a new adversarial model, which we call the τ -Stream Adversary Model, formally introduced in Definition 24. This adversarial model, parameterized by an integer $\tau \geq 1$, is very much similar to our main bounded-memory model (see Definition 11), but we allow for it to have unbounded persistent memory, and it can thus memorize the entire interaction history with the streaming algorithm. To counter its power, we only allow it to switch (arbitrarily) between τ streams of pre-generated updates as it interacts with the streaming algorithm. For this adversary, we show how to turn a correct order-independent oblivious streaming algorithm into one which can handle the τ -Stream Adversary Model at the cost of, roughly, a *linear* in τ blow-up in space. At a high level, the order-independent property allows us to union bound over the selection of prefixes of the initial τ streams, of which there are only $O(m^\tau)$. Using the standard median of estimates trick, this implies that our overall space dependence is logarithmic in the number of options, resulting in the claimed linear in τ blow-up. Note that this is reminiscent of the computational paths technique [5]. See Lemma 26 for a formal argument.

This intermediate τ -Stream Adversary Model is useful because we can reduce the randomized adversary in the bounded-memory model to a τ -Stream Adversary for an appropriate value of τ . More formally, the proof of Theorem 27 gives a reduction from the randomized adversary with k bits of persistent memory to the τ -Stream Adversary Model for $\tau = O(2^k \epsilon^{-1} \log \alpha)$. At a high level, this comes from the fact that the persistent memory can take on at most 2^k different states and the total number of “different” estimates of the streaming algorithm can be assumed to be $O(\epsilon^{-1} \log \alpha)$, via a rounding argument (see Lemma 13). Hence, we can view the adversary with k bits of persistent memory as interleaving between $\tau = O(2^k \epsilon^{-1} \log \alpha)$ many pre-defined streams. See the proof of Theorem 27 for details. Lastly, we remark that Theorem 27 has downstream corollaries to order-independent problems, including moment estimation. See Section 6.4 for details.

2 Preliminaries

For any $k \in \mathbb{N}$, we write $[k] \stackrel{\text{def}}{=} \{1, 2, \dots, k\}$ to denote the set of the k smallest positive integers. We focus on problems in which the exact solution is a single number. Since we consider these problems in the streaming setting, it is usually not possible to solve them exactly in small space and we aim to solve them approximately, staying within a multiplicative factor close to 1 with respect to the exact solution. This notion of approximation is defined next.

► **Definition 4** (Multiplicative approximation). *Let $\epsilon, x, y \in [0, \infty)$. We say that y is a $(1 + \epsilon)$ -multiplicative approximation to x if $x \leq y < (1 + \epsilon)x$.*

Note that we used here for convenience a one-sided notion of approximation, where we do not allow the approximate value y to be smaller than x . However, by standard shifting arguments, all results in this paper would hold, with small adjustments, if we allowed for error in both directions, i.e., if we required that $x/(1 + \epsilon) < y < (1 + \epsilon)x$ (and/or if we replace the strict inequality signs with $\leq$).

We now introduce the general form of problems in which we are interested.

► **Definition 5** (Estimation problems). *For a given function $f : \mathbb{Z}^n \rightarrow \{0\} \cup [1, \alpha]$ and parameter $\epsilon > 0$, we refer to the task of computing a $(1 + \epsilon)$ -multiplicative approximation to $f(v)$ on a given input $v \in \mathbb{Z}^n$ as the (f, ϵ) -estimation problem. (To simplify our presentation, we require that $\alpha \geq 2$ so that $\log \alpha \geq \log 2 > 0$, i.e., $\log \alpha$ is not less than some positive constant.)*

We usually do not have an explicitly given vector v that is an input to an (f, ϵ) -estimation problem and instead we work with a stream of updates to the initial all-zero vector.

► **Definition 6** (Frequency vectors and streams of updates). *We refer to inputs of our estimation problems as frequency vectors. For a given vector $v \in \mathbb{Z}^n$, v_i is the frequency or number of occurrences of item $i \in [n]$. (Note that we allow frequencies to be negative.)*

A frequency vector $v \in \mathbb{Z}^n$ can be defined via a stream of updates, which is a sequence of $t \in \mathbb{N}$ pairs of the form $(i_j, \Delta_j) \in [n] \times \{-1, 1\}$ for $j \in [t]$. The first element of the pair, i_j , describes the element whose frequency is being updated. The second element describes whether the update decreases (when $\Delta_j = -1$) or increases (when $\Delta_j = 1$) the frequency of i_j . We refer to pairs (i_j, Δ_j) as updates.

A stream of updates of this form – or, in fact, any prefix of it – accumulates to a frequency vector. For a stream of t updates as above, for any $j \in \{0\} \cup [t]$, we write $v^{(j)}$ to denote the frequency vector in $\mathbb{Z}^n$ obtained by applying the first j updates to the all-zero vector. Formally, for all $i \in [n]$,

$$v_i^{(j)} \stackrel{\text{def}}{=} \sum_{\substack{1 \leq j' \leq j \\ i_{j'} = i}} \Delta_{j'}. \quad (1)$$

We now define (a version of) the standard streaming model in which the stream is fixed in advance.

► **Definition 7** (“Oblivious” streaming). *A streaming algorithm in the oblivious adversary model (or, for short, oblivious streaming algorithm) for an (f, ϵ) -estimation problem is given the parameter ϵ and an upper bound m on the length of the stream. Then, after receiving one by one $t \leq m$ updates (i_j, Δ_j) for $j \in [t]$, it produces a solution to the (f, ϵ) -estimation problem on $v^{(t)}$, which is the cumulative frequency vector defined in Definition 6.*

We now introduce the notion of *order-invariant* oblivious streaming algorithms. Intuitively, the definition is relevant to problems in which the current value of f on the underlying stream depends only on the current frequency vector – $v^{(j)}$ at round j – and not on the *order* of updates that accumulated to $v^{(j)}$. We note that similar concepts have been defined before – e.g., “path independent automata” [22] – but we state the following version for completeness. Order-invariant algorithms are central in the streaming literature, since linear sketches are order-invariant.

► **Definition 8** (Order invariant streaming algorithms [22]). *Let $\mathcal{A}$ be an oblivious streaming algorithm for an (f, ϵ) -estimation problem that processes streams $S = ((i_1, \Delta_i), \dots, (i_t, \Delta_t))$ of at most $t \leq m$ updates. Furthermore, suppose that $\mathcal{A}$ uses an internal random seed $\rho \in \{0, 1\}^*$ and that its memory after processing an entire stream S of updates for a random seed ρ is $\mathcal{M}(S, \rho)$. Let $v(S)$ denote the frequency vector that is the result of accumulating all updates in S . (This corresponds to $v^{(t)}$ in Definition 6 for the given stream of updates of length t).*

We say that $\mathcal{A}$ is order invariant if for any pair S_1 and S_2 of streams of length at most m such that $v(S_1) = v(S_2)$ and any possible random seed ρ , we have $\mathcal{M}(S_1, \rho) = \mathcal{M}(S_2, \rho)$.

We conclude by formally defining two parameters of streams, *flip number* and *density*, which have both proved important in the robust streaming literature. Indeed, most of the existing robust streaming frameworks have space complexity which depends on the flip number, and, in some cases, the density as well.

► **Definition 9** (Flip number [5]). *Let $\epsilon \geq 0$, $m \in \mathbb{N}$, and $\bar{y} = (y_0, y_1, \dots, y_m)$ be any sequence of real numbers. The ϵ -flip number of $\bar{y}$ is the maximum $k \in \mathbb{N}$ such that there exists $0 \leq i_1 < \dots < i_k \leq m$ so that $y_{i_j-1} \notin [(1 - \epsilon)y_{i_j}, (1 + \epsilon)y_{i_j}]$ for every $j = 2, 3, \dots, k$.*

In our paper, the value of ϵ when defining the flip number will be clear from context.

► **Definition 10** (Density/sparsity of a stream). *For a vector $x \in \mathbb{R}^n$, we let $\|x\|_0 \stackrel{\text{def}}{=} |\{x_i \mid x_i \neq 0\}|$. We refer to this quantity as the density (sometimes also referred to as the sparsity) of vector x . The density (or sparsity) of a stream of updates at time j is the density of the frequency vector $v^{(j)}$, where $v^{(j)}$ is as defined in Definition 6.*

3 The Bounded-Memory Adversary Model

Our adversarial streaming setting is formalized in the following definitions. It builds on and is inspired by the definition of robust streaming, which was introduced by Ben-Eliezer, Jayaram, Woodruff, and Yaguev [5]. The main difference here is that we carefully restrict the adversary’s memory.

The model describes a game between two players, an adversary, which we often call **Adversary**, and a streaming algorithm, which we often call **Algorithm**, over m rounds. For every round $j \in [m]$, the goal of **Algorithm** is to output an estimate y_j to the value of a fixed function f (i.e., it is a *tracking* streaming algorithm, as opposed to the algorithm in Definition 7, which only outputs an estimate once, at the end of the stream).

Formally, we fix an integer $\alpha \geq 2$, let $\epsilon > 0$ be an approximation parameter, and let function $f : \mathbb{Z}^n \rightarrow \{0\} \cup [1, \alpha]$ be a function of the underlying frequency vector as defined in Equation (1). As a concrete example, f could output the F_p -moment of the frequencies or the number of distinct elements. We assume both parties know f , α , and ϵ . The *adversarial streaming model* is formalized as follows:

► **Definition 11** (Bounded-Memory Adversary Model). *The Bounded-Memory Adversary Model is a game between two players, an **Adversary** = $\text{Adversary}(r, k, \mathcal{P}, \mathcal{E}, \mathcal{W})$ and an **Algorithm**. $\mathcal{P}, \mathcal{E}$, and $\mathcal{W}$ are different types of memory accessible to **Adversary**, defined shortly. Initially, they are all blank. The game proceeds in m rounds, corresponding to values of $j \in [m]$ from 1 through m , according to the following rules. For $j \geq 2$:*

- **Adversary** draws fresh randomness $r_j \in \{0, 1\}^*$ with read-only access. After computing the update (i_j, Δ_j) (see next bullet), the memory containing r_j is made blank.
- **Adversary** computes an update $(i_j, \Delta_j) \in [n] \times \{-1, 1\}$, which depends only on the randomness r_j , a persistent memory $\mathcal{P}$ of at most k bits, and a read-only estimate memory $\mathcal{E}$ containing an estimate $y_{j-1} \in \{0, 1\}^*$. **Adversary** has both read and write access to $\mathcal{P}$ (and, in particular, can write to $\mathcal{P}$ during its computation of (i_j, Δ_j)), but only read access to $\mathcal{E}$. When computing (i_j, Δ_j) , **Adversary** also has read and write access to an unbounded working memory $\mathcal{W}$. After computing update (i_j, Δ_j) , the working memory $\mathcal{W}$ is made blank.
- **Algorithm** receives (i_j, Δ_j) , updates its internal memory, and produces an estimate y_j .
- The estimate memory $\mathcal{E}$ of **Adversary** is then overwritten with y_j .

For $j = 1$, the update (i_j, Δ_j) is computed analogously, but it does not depend on any previous estimates (as there are none).

Let $v^{(j)}$, for $j \in [m]$, be the frequency vector accumulating the first j updates as in Definition 6. If in all rounds $j \in [m]$, y_j is a correct solution to the (f, ϵ) -estimation problem on $v^{(j)}$, then we say that **Algorithm** succeeds. Otherwise, **Algorithm** fails. Finally, we say that **Adversary** is deterministic if the random seeds are always set to the empty string.

Note that the above definition only specifies the restrictions on **Adversary** and does not explicitly restrict the space used by **Algorithm**. Our theorems provide restrictions on the space required by **Algorithm**, which is the main computational resource optimized in streaming algorithms.

Recall that the estimate memory $\mathcal{E}$ models the fact that the adversary has access to the previous output of **Algorithm**. This easily extends to the case that **Adversary** has access to any fixed number t of previous outputs by specifying that the estimate memory $\mathcal{E}$ contains the last t estimates output by **Algorithm**. All results proved in this paper extend in a straightforward manner to this setting.

► **Remark 12** (Discussion of randomness: fresh vs. persistent). Note that our model lets the adversary access fresh randomness in each round. Since the adversary has limited amount of memory, storing outcomes of coin tosses across several rounds may not be possible. A natural question in this setting is whether access to a persistent infinite random string, drawn before the interaction with the algorithm, could make the adversary more powerful and more difficult to defend against. After all, the random string could, for instance, be used to simulate a random oracle, which could then provide a fully independent hash function with no extra storage needed.

It turns out that this would *not* make designing adversarially robust algorithms more difficult. Our goal is to provide robust algorithms that output correct estimates throughout the entire stream with probability at least $1 - \delta$, where δ is a parameter selected by the user of the algorithm. It is not difficult to show (see the full version of the paper) that if there is an adversary who succeeds at breaking the estimates with probability greater than δ , using persistent randomness, a successful configuration of persistent randomness can be baked into its internal logic, which makes it an adversary in our model with no persistent randomness.

Conversely, one could ask whether fresh randomness is needed, because it may seem that the initial persistent randomness could be used to simulate it. This would be true if the adversary had enough memory to keep track of which bits of persistent randomness have

already been used. For instance, this could be achieved by maintaining the pointer to the first unused bit of randomness. Our results are meaningful only if the adversary has access to a very small amount of persistent memory. In this setting, after the persistent randomness is fixed, the adversary generates the same update for a given estimate in the same state of memory. This makes it a deterministic adversary, which is easier to analyze, because we can force it to produce a low-sparsity frequency vector by rounding the algorithm's estimates. More details on how this is achieved are provided when we discuss our results for deterministic adversaries.

4 Deterministic Adversaries

In this section we prove our robust streaming result against deterministic bounded-memory adversaries. First, the following lemma shows that the possible outputs of f admit a small “ ϵ -net” with respect to multiplicative approximation. We defer its straightforward proof to the full version of the paper.

► **Lemma 13.** *Consider the interval $[1, \alpha]$ for an integer α that satisfies $\alpha \geq 2$. Let $\epsilon \in (0, 1)$. There exists a subset $\mathcal{N} \subseteq [1, \alpha]$ of size $O(\epsilon^{-1} \log \alpha)$ with the property that*

$$\forall x \in [1, \alpha], \quad \exists y \in \mathcal{N} \text{ satisfying } x \leq y < (1 + \epsilon)x.$$

Our main result provides efficient streaming algorithms against deterministic memoryless adversaries. It is assumed that we are able to compute f and round its value to the net produced by Lemma 13 using $M(f, \epsilon)$ bits by just specifying the non-zero coordinates of the input vector.

► **Theorem 14.** *Consider the Bounded-Memory Adversary Model (see Definition 11) with a deterministic adversary and $k = 0$, i.e., no persistent storage. There exists a deterministic streaming algorithm that always succeeds and uses $O\left(\frac{\log(\alpha)}{\epsilon} \cdot \log(mn) + M(f, \epsilon)\right)$ bits of space.*

Proof. We construct a deterministic algorithm, **Algorithm**, and analyze its correctness and space complexity. Its description is as follows:

1. The algorithm maintains a *sparse* representation of the *exact* frequency vector v , which equals $v^{(j)}$ at time j . More specifically, it explicitly keeps a mapping from all coordinates $i \in [n]$ that have received at least one update to their current frequencies. This is possible because the frequencies of all other coordinates equal 0. Initially, the frequency vector v is the all zero vector – $v \leftarrow (0, 0, \dots, 0)$ – so the mapping is empty.
2. At any round $j \in [m]$, upon receiving an update (i_j, Δ_j) , it simply updates the frequency of coordinate i_j in the mapping it maintains by setting it to $v_{i_j} \leftarrow v_{i_j} + \Delta_j$. If the mapping from i_j is not present in the mapping, it is first added with the initial frequency of 0.
3. To produce estimate y_j , **Algorithm** computes and outputs a rounded version of the value $f(v^{(j)}) \in \{0\} \cup [1, \alpha]$. This process can be seen as first computing $f(v)$, where $v = v^{(j)}$ is the current frequency vector, and then rounding it up to the smallest element that is at least $f(v)$ in the net $\mathcal{N} \cup \{0\}$ given by Lemma 13, before outputting it. Thus, we output $y_j \in \mathcal{N} \cup \{0\}$ satisfying $f(v^{(j)}) \leq y_j < (1 + \epsilon)f(v^{(j)})$ and $y_j \in \mathcal{N}$. We know that this can be achieved, using at most $M(f, \epsilon)$ space.

The correctness of **Algorithm** follows from the fact that it can always calculate the *exact* value of $f(v^{(j)})$ and round it up to a value y_j in the net $\mathcal{N} \cup \{0\}$ from Lemma 13, which by definition is a $(1 + \epsilon)$ -multiplicative approximation to $f(v^{(j)})$. For the space complexity, note

that **Adversary** sees at most $|\mathcal{N}| + 1 = O(\epsilon^{-1} \log(\alpha))$ unique responses from **Algorithm**. Since **Adversary** is deterministic (and its persistent memory and random string are always blank), it follows that it is only a function of its estimate memory $\mathcal{E}$, which can only take on at most $|\mathcal{N}| + 1$ different values. Thus, $|\{i_1, \dots, i_m\}|$, where i_j is the update of **Adversary** at round j , is bounded by $|\mathcal{N}| + 1 + 1 = O(\epsilon^{-1} \log(\alpha))$, where the additional additive term of 1 comes from the fact that, initially, the adversary produces a (potentially different) single update that does not depend on any estimate, because there is no estimate initially. That is, the total number of unique items that the adversary places in the stream (either for insertions or deletions) is also at most $O(\epsilon^{-1} \log(\alpha))$. Thus, the total number of coordinates that **Algorithm** ever has to maintain is also bounded by the same quantity, leading to the claimed space bound. Note the $\log m$ term in the $\log(mn) = \log m + \log n$ overhead comes from the fact that a single frequency that **Algorithm** maintains could be any integer value between $-m$ and m . The $\log n$ term comes from the fact that we also need to store the elements in $[n]$ that **Algorithm** has encountered. This completes the proof. $\blacktriangleleft$

The proof easily extends to the case of $k > 0$, when **Adversary** has k bits of persistent memory. In this case, **Adversary**'s outputs (which are the stream updates) are a deterministic function of at most $O(\epsilon^{-1} \log \alpha) \cdot 2^k$ different possible values.

► **Corollary 15.** *Consider the Bounded-Memory Adversary Model (Definition 11) with a deterministic adversary and persistent memory of k bits. There exists a deterministic streaming algorithm that always succeeds and uses $O\left(2^k \cdot \frac{\log(\alpha)}{\epsilon} \cdot \log(mn) + M(f, \epsilon)\right)$ bits of space.*

5 Memoryless Adversaries Can Generate Non-Trivial Streams

In this section we prove the following result, regarding the ability of memoryless or extremely low-memory adversaries to generate streams that are both non-sparse and have large flip number.

► **Theorem 16.** *Fix $0 < c < 1/2$. Let $m, n \in \mathbb{N}$ and $\epsilon \in (0, 1]$,⁵ where $n > 10m^{2c}$. Consider an instantiation of the Bounded-Memory Adversary Model for the (F_2, ϵ) -estimation problem. There exists a randomized memoryless **Adversary** that, conditioned on **Algorithm** correctly solving the (F_2, ϵ) -estimation problem over the entire stream, with probability at least $9/10$, generates a stream of length m whose density is $\Omega(m^c)$ for all but the first m^c rounds of the stream, and whose flip number is $\Omega(m^{1-c})$.*

*In addition, if we provide **Adversary** with one bit of persistent memory (i.e., $k = 1$ in Definition 11), then the above statement holds, for any constant $0 < c < 1$, with flip number $\Omega(m^{1-0.5c})$.*

Proof. We start with the proof of the second part of the theorem statement, in which **Adversary** has *one bit* of persistent memory. The strategy and proof required for a memoryless adversary are slightly more complicated, and we discuss them later in the proof.

For convenience, we assume that m^c is an integer; if not, we can replace it with $\lceil m^c \rceil$ as needed.

⁵ Note that we allow ϵ to be arbitrarily small. In particular, it can depend on parameters such as m and n .

When Adversary has one bit of persistent memory. Consider the following adversarial strategy requiring one bit of persistent memory. **Adversary** uses the persistent memory bit to track in which of two possible states it is. These states are $\uparrow$ (“going up”) and $\downarrow$ (“going down”). The initial state is $\uparrow$.

- **Type I insertion:** If the previous F_2 estimate sent by **Algorithm** is smaller than m^c (or if it is the first round of the game and no previous estimate is available), **Adversary** picks a uniformly random element i from $\{2, 3, \dots, n\}$ and *inserts* i . The state of **Adversary** is set to $\uparrow$.
- **Type II insertion:** If the previous F_2 estimate of **Algorithm** is at least m^c and less than $(1 + \epsilon)m^c$, **Adversary** *inserts* the element 1 and sets its state to $\uparrow$.
- **Deletion:** If the previous F_2 estimate of **Algorithm** is at least $(1 + \epsilon)^3 m^c$, **Adversary** *deletes* the element 1 and sets the state to $\downarrow$.
- **Following the sign:** If the previous F_2 estimate is at least $(1 + \epsilon)m^c$ and less than $(1 + \epsilon)^3 m^c$, then **Adversary** *inserts* 1 if its internal state is $\uparrow$, and *deletes* 1 if its internal state is $\downarrow$. The internal state remains unchanged after this operation.

Note that **Adversary** can be implemented in the Bounded-Memory Adversary Model with one bit of persistent memory. Indeed, the action of **Adversary** at any given time depends only on the following three sources: (i) the previous estimate of the algorithm, (ii) whether the previous state is $\uparrow$ or $\downarrow$, and (iii) a random sample from the uniform distribution on $\{2, \dots, n\}$. It remains to prove that this adversarial strategy satisfies the statement of the theorem. In the following, we write $v^{(t)}$ to denote the frequency vector after t updates, i.e., we use the notation from Definition 6.

We now prove a sequence of claims about the nature of the game given the above adversarial strategy. We stress that these claims hold *for any Algorithm* (even one that has unlimited memory), as long as **Algorithm** correctly solves the (F_2, ϵ) -estimation problem after each update.

▷ **Claim 17.** There are at most m^c Type I insertions throughout the stream.

Proof. Once m^c insertions of this type have been made, the actual value of F_2 remains at least m^c , because elements from $\{2, 3, \dots, n\}$ are never deleted. By the correctness of estimates, the estimate visible to **Adversary** must also be at least m^c . This implies that no Type I insertion can occur in subsequent rounds. ◁

Consider the event that no element from $\{2, 3, \dots, n\}$ was inserted twice throughout the stream; let us call this event *unique insertion*. Note that these elements are only added via Type I insertions, and are never deleted. As discussed above, at most m^c type I insertions can occur throughout the stream. Via a standard birthday paradox argument, unique insertion happens with probability at least $19/20$ if $n \geq 10(m^c)^2 = 10m^{2c}$.

For the rest of the proof, we condition on the unique insertion event (which, as mentioned, occurs with probability at least $19/20$). Assuming unique insertion, the sparsity part of the theorem easily follows.

▷ **Claim 18.** After the first m^c rounds of the adversarial game (and throughout the rest of the stream), the density of the stream is always at least $m^c/(1 + \epsilon) = \Omega(m^c)$.

Proof. As long as the density of the stream is at most $m^c/(1 + \epsilon)$, **Algorithm** must return an output smaller than m^c . Thus, in the first $1 + \lfloor m^c/(1 + \epsilon) \rfloor$ rounds of the stream, all actions by **Adversary** are Type I insertions. These inserted elements are all distinct, assuming unique insertion, and are never deleted. Thus the density remains at least $m^c/(1 + \epsilon)$ for the rest of the game. ◁

The rest of the proof establishes the flip number lower bound (which occurs with probability 1, conditioned on unique insertion). We start with the following claim.

▷ **Claim 19.** The frequency of 1 in the stream, at any point throughout the game, is non-negative.

Proof. Suppose that this is not the case. Then there exists some minimal time t after which the frequency of 1 becomes negative for the first time. Thus, at round $t + 1$, **Adversary** chooses to delete 1, meaning that we are in either the “Deletion” or “Following the sign” regime. This, in turn, implies that **Algorithm**’s estimate at time t is at least $(1 + \epsilon)m^c$.

On the other hand, the frequency of 1 at time t must be zero (since it turns negative after the deletion). By Claim 17 and unique insertion, $F_2(v^{(t)}) \leq m^c$. By correctness, **Algorithm**’s estimate at time t must be less than $(1 + \epsilon)m^c$. This stands in contradiction to the first paragraph. ◁

Let S denote the set of all times throughout the stream at which a Type I insertion is made. By Claim 17, $|S| \leq m^c$. Our last two claims of the proof are given next.

▷ **Claim 20.** Let $t \leq m - (1 + \epsilon)^4 m^c$ and suppose that (i) $F_2(v^{(t)}) \leq (1 + \epsilon)m^c$ and (ii) the internal state of **Adversary** at time t is $\uparrow$.

Then there exists $t' \in [t + 1, m]$ such that (i) $F_2(v^{(t')}) > (1 + \epsilon)^2 m^c$, (ii) the internal state of **Adversary** is $\downarrow$ at time t' and $\uparrow$ at time $t' - 1$, and (iii) the number of values $t'' \in [t, t']$ not belonging to S is at most $O(m^{c/2})$.

▷ **Claim 21.** Let $t \leq m - (1 + \epsilon)^2 m^{c/2}$ and suppose that (i) $F_2(t) > (1 + \epsilon)^2 m^c$ and (ii) the internal state of **Adversary** at time t is $\downarrow$, and at time $t - 1$ it is $\uparrow$. There exists a time t' such that $t < t' < t + O(m^{c/2})$, $F_2(v^{(t')}) \leq (1 + \epsilon)m^c$, and the internal state at time t' is $\uparrow$.

With the above two claims in hand, one can see that the flip number of the stream is $\Omega(m^{1-c/2})$. Indeed, when disregarding all rounds $t \in S$ of the game (note that there are only $O(m^c)$ such rounds), the above two claims show that the output of the algorithm must flip – more specifically, either increase or decrease by a factor of $1 + \epsilon$ – every $O(m^{c/2})$ rounds, and so the flip number is at least

$$\frac{m - O(m^c)}{O(m^{c/2})} = \Omega(m^{1-\frac{c}{2}}).$$

We now prove these claims.

Proof of Claim 20. Consider any time t satisfying the conditions of the claim. Let $t' > t$ be the minimal integer for which (i) $F_2(v^{(t')}) > (1 + \epsilon)^2 m^c$, (ii) the internal state of **Adversary** at time t' is $\downarrow$, and at time $t' - 1$ it is $\uparrow$. We note that such t' must exist as long as $t \leq m - (1 + \epsilon)^4 m^c$, since all updates between t and t' are insertions and all entries of the frequency vector at time t are non-negative.

For any $t'' \notin S$ such that $t \leq t'' < t'$, the action taken by the adversary in round $t'' + 1$ is either a Type II insertion or a “Follow the Sign” step with an $\uparrow$ state (both of which lead to inserting 1). Indeed, if $F_2(v^{(t'')}) \leq (1 + \epsilon)^2 m^c$, then the correctness of **Algorithm** implies that the estimate at time t'' is smaller than $(1 + \epsilon)^3 m^c$; and otherwise, the minimality of t' implies that the sign at time t'' must be $\uparrow$. In both cases, the internal state remains $\uparrow$, as the only action that changes it to $\downarrow$ is a deletion.

Finally, $\Theta(m^{c/2})$ insertions of 1 suffice to increase the F_2 value of the stream by $\Omega(m^c)$, thus reaching t' as above within $O(m^{c/2})$ rounds (disregarding rounds with Type I insertions). ◁

Proof of Claim 21. From the condition that the sign at time $t - 1$ is $\uparrow$, we know that $F_2(v^{(t-1)}) \leq (1 + \epsilon)^3 m^c$. In particular, the contribution of the element 1 to $F_2(v^{(t)})$ is $O(m^c)$, meaning that the frequency of 1 at time t is $O(m^{c/2})$. Let $t' > t$ denote the first time at which an insertion of Type I or Type II occurs. Note that such a time t' must exist, provided $t < m - O(m^{c/2})$, and in addition, $t' < t + O(m^{c/2})$. Indeed, as long as we only have deletions or follow the sign rounds, the internal state remains $\downarrow$, leading to deletions of the element 1. Now, because the frequency of 1 is $O(m^{c/2})$ at time t , after $O(m^{c/2})$ rounds we reach a situation in which either a type I or II insertion happens, or the frequency of 1 becomes zero, in which case the F_2 value is at most m^c and the next step must either be a type I or II insertion (depending on the estimate output by **Algorithm**).

Now, since **Adversary** made a type I or type II insertion at time t' , the estimate of **Algorithm** at time t' must be at most $(1 + \epsilon)m^c$. Thus, the actual F_2 value is also at most $(1 + \epsilon)m^c$. The proof follows. $\triangleleft$

When Adversary is memoryless. The strategy of **Adversary**, as well as the analysis, are quite similar to the above case. We detail below the modifications required in the strategy and proof.

- **Type I insertions, Type II insertions, and deletions** remain the same, except that that the deletion regime only applies if the previous estimate is at least $(1 + \epsilon)^4 m^c$ (not $(1 + \epsilon)^3 m^c$ as previously defined). In addition, we now no longer have the internal state (of $\uparrow$ or $\downarrow$). Note that in all of these three action types, the action chosen by the adversary with one bit of permanent memory did not depend on the internal state, and so these remain well-defined.
- The “follow the sign” action is replaced with an **(unbiased) random walk** action: If the previous F_2 -estimate is at least $(1 + \epsilon)m^c$ and less than $(1 + \epsilon)^4 m^c$, then **Adversary** inserts 1 with probability $1/2$, and deletes 1 with probability $1/2$.

This strategy by **Adversary** is clearly memoryless: each step depends only on the last estimate provided by **Algorithm**, where in particular **Adversary** may sample the action from a fixed distribution (depending on the last estimate). Claims 17, 18, and 19 still hold (and their proofs remain unchanged), again conditioning on the unique insertion event, which holds with probability at least $19/20$. It remains to prove the flip number lower bound.

We next give a result analogous to Claims 20 and 21 in the memoryless case. Its proof relies on standard hitting time results on random walks in one dimension, together with a simple coupling argument.

▷ **Claim 22.** There exists an absolute constant $\alpha > 0$ satisfying the following. For every t such that $0 \leq t \leq m - \alpha \cdot m^c$, there exists a t' such that $t < t' \leq t + \alpha \cdot m^c$, $F_2(v^{(t')}) \notin (1 \pm \epsilon)F_2(v^{(t)})$, with probability at least $1/2$, independently of all randomness revealed before round t of the game.

We now prove Claim 22. For the proof, we use a standard hitting time result on random walks in one dimension. An unbiased random walk in 1D refers to the following process: start at location 0 on the one-dimensional grid. In each round, move one step to the right (i.e., from location s to $s + 1$, where s is the current location) with probability $1/2$ (independently of all previous steps), and one step to the left (i.e., from s to $s - 1$) otherwise. The hitting time of the random walk with respect to a value s is the first round in which the random walk arrives at location s .

► **Lemma 23** (Hitting time of 1D random walk; see, e.g., [1], Chapter 5). *The hitting time for any location $s \in \mathbb{Z} \setminus \{0\}$ in an unbiased 1D random walk is with probability $1/2$ at most βs^2 , where $\beta > 0$ is an absolute constant.*

Proof of Claim 22. There are two cases that one should consider. The first case is $F_2(v^{(t)}) < (1 + \epsilon)^2 m^c$, and in this case we prove the statement of the claim for some t' satisfying that $F_2(v^{(t')}) \geq (1 + \epsilon)^3 m^c > (1 + \epsilon)F_2(v^{(t)})$. The second is the complementary case, in which $F_2(v^{(t)}) \geq (1 + \epsilon)^2 m^c$, and we prove the analogous statement for some $t' < (1 + \epsilon)m^c \leq F_2(v^{(t)})/(1 + \epsilon)$. These cases are very similar, as we shall explain soon. For now, let us consider the first case.

Consider a coupling of the adversarial game to a random walk, where each insertion of 1 is viewed as a step up, and each deletion of 1 is a step down (and we think of insertions of elements other than 1 as not changing the state). By Lemma 23, with probability at least $1/2$ an unbiased random walk will hit location $(1 + \epsilon)^{3/2} m^{c/2}$ in time $O((m^{c/2})^2) = O(m^c)$. Now what about the adversarial game, starting at time t , and as long as we do not exceed the F_2 value of $T \stackrel{\text{def}}{=} (1 + \epsilon)^3 m^c$? It turns out the the frequency of 1 in this case is not an unbiased random walk, but it actually dominates the unbiased case: The probability to step up can be 1 in some cases, while the probability to step down is never higher than $1/2$. Indeed, in this regime there are three possible actions: type I and type II insertions, and random walk operations.

There are at most m^c rounds in total in which type I insertions occur. Type II insertions always lead to an insertion, i.e., a step up. Random walk steps lead to an insertion of 1 with probability $1/2$ and deletion with probability $1/2$. Thus, we see that in all but $O(m^c)$ rounds (in which type I insertions occur, which do not change the frequency of the element 1), the adversarial game dominates an unbiased random walk, as long that the F_2 value has not exceeded T . This implies that with probability $1/2$, one of the following must hold between time t and $t + O(m^c)$: either we reach an F_2 value of at least $T = (1 + \epsilon)^3 m^c$, leading to a flip (with respect to time t), or the frequency of 1 in the adversarial game reaches $(1 + \epsilon)^{3/2} m^{c/2}$, making the F_2 value reach T and causing a flip again. This concludes the first case.

The analysis of the second case is completely symmetric, except that we set the threshold at $T \stackrel{\text{def}}{=} (1 + \epsilon)m^c$, and there are only two possible operations: either a deletion (leading to a step down with probability 1) or a random walk action. In both cases, the locations is dominated by an unbiased random walk, and again by an application of Lemma 23, the statement of the claim follows. ◁

With the claim in hand, it is straightforward to show an $\Omega(m^{1-c})$ flip number lower bound. Indeed, consider a sequence of all values of t which are multiples of $\lfloor \alpha \cdot m^c \rfloor$ between 1 and m ; there are $\Omega(m^{1-c})$ such values. The probability of a flip between each consecutive pair of t values in this sequence is at least $1/2$, independently of all previous information, and so the expected number of flips is $\Omega(m^c)$. It follows, e.g., from Chernoff bound, that the total number of flips will be at least half its expectation with probability $1 - 2^{-\Omega(m^c)} > 0.95$, where the inequality holds for large enough m . This completes the proof. ◀

6 Randomized Adversaries

6.1 The τ -Stream Adversary Model

As discussed before, in the oblivious adversary model, the adversary selects the entire stream in advance – without any interaction with the streaming algorithm – and is only required to provide a correct estimate at the end of the stream. There is a standard way to turn a

correct algorithm in the oblivious setting into an algorithm with a *tracking* guarantee (i.e., an algorithm that provides a correct estimate at any point along the stream) as long as the adversary remains oblivious. The standard approach is first to amplify the probability of the algorithm's success at any point, by using the median of estimates from a small number of copies of the algorithm. Then one applies the union bound to claim that the algorithm is unlikely to err at any point.

We now consider a generalization of the oblivious adversary model, which we call the τ -Stream Adversary Model. This model introduces adaptivity by allowing the adversary to select τ streams in advance (obliviously) and arbitrarily (and adaptively) merge them into the actual stream passed to the algorithm. We describe it in relation to the Bounded-Memory Adversary Model, introduced in Definition 11. This time, **Adversary** is allowed to use an unlimited amount of memory, but it is a great question if restricting it could lead to better robust algorithms in this model (see Section 7).

► **Definition 24** (τ -Stream Adversary Model). *Let τ be a positive integer. The τ -Stream Adversary Model is a game between two players, **Algorithm** and **Adversary**, with the same interaction scheme as in the Bounded-Memory Adversary Model: in each round, **Adversary** sends an update and **Algorithm** replies with an estimate. As opposed to the Bounded-Memory Adversary Model, **Adversary** is allowed unlimited persistent memory, but is restricted in its choice of the stream it generates as follows:*

1. *Initially, before the first round, **Adversary** generates the first update, to be used in round 1. In addition, **Adversary** generates τ streams of $m - 1$ updates each at this point.*
2. *In rounds 2 through m , **Adversary** can use the complete knowledge it has collected thus far, including the latest estimate provided by **Algorithm**, to select one of the τ streams arbitrarily. Then **Adversary** removes the first element of that stream and passes it to **Algorithm** as the next update.*

Note that thanks to the unlimited persistent memory, **Adversary** can record the entire history of stream updates and algorithm outputs. The only restriction on **Adversary** is the fact that it must draw the next update at any time from one of its τ fixed streams.

6.2 Robust Algorithms for the τ -Stream Adversary Model

We now describe an efficient algorithm for this model, in which we apply the standard median trick to increase the probability of success of a streaming algorithm in the oblivious setting. This is captured in the following standard lemma, whose proof is included in the full version of the paper.

► **Lemma 25** (Standard amplification trick). *Let $\mathcal{A}$ be an oblivious streaming algorithm that, with probability at least $9/10$, produces a correct solution to a (f, ϵ) -estimation problem. One can obtain an oblivious streaming algorithm that provides a correct solution to this problem with probability $1 - \delta$, for any $\delta \in (0, 1/10)$, by running $t \stackrel{\text{def}}{=} \lceil 12 \ln(1/\delta) \rceil$ independent copies of $\mathcal{A}$ and outputting the median of their estimates.*

We use the above lemma to prove that there is a robust algorithm for the τ -Stream Adversary Model, whose space complexity scales linearly with τ (so it is efficient when τ is small). Here we need the additional condition that $\mathcal{A}$ is *order-invariant* (see Definition 8). More specifically, we show that an order-invariant oblivious streaming algorithm is very likely to provide correct estimates throughout the entire stream if its probability of success has been amplified so that it errs with probability much smaller than $m^{-\tau}$.

► **Lemma 26.** *Let $\mathcal{A}$ be an order-independent oblivious streaming algorithm that uses $M(\epsilon, \delta)$ space to provide a correct solution to an (f, ϵ) -estimation problem with probability $1 - \delta$ for any $\epsilon \in (0, 1)$ and $\delta \in (0, 1)$. There is a robust streaming algorithm for the same problem in the τ -Stream Adversary Model that uses $M(\epsilon, \delta/m^\tau)$ space, where $\epsilon \in (0, 1)$ and $\delta \in (0, 1)$, and succeeds with probability at least $1 - \delta$. The space of the robust algorithm can also be bounded by $O(\tau \log m + \log(1/\delta)) \cdot M(\epsilon, 1/10)$.*

Proof. The only streams on which **Algorithm** may be asked to provide an estimate are a result of merging prefixes of the initial τ streams and appending to the first update. Both the initial τ streams and the first update are selected without any interaction with **Algorithm**, i.e., non-adaptively. Note that there are at most $t \stackrel{\text{def}}{=} m^\tau$ ways in which prefixes of the τ streams can be selected to amount to streams of length between 1 and m , which are the lengths on which **Algorithm** has to provide estimates. While the selected prefixes can potentially be merged in many different ways into a stream that **Algorithm** processes, $\mathcal{A}$ is order-independent and it therefore behaves in an exactly the same way for a given selection of prefixes, independently of the order in which it processes the updates they contain. Hence, to leverage $\mathcal{A}$, it suffices to ensure that it provides a correct estimate on one arbitrary ordering of elements in the selected prefixes to provide a correct ordering for all other orderings. Furthermore, it suffices that $\mathcal{A}$ produces a correct estimate for each selection of prefixes of the initial τ streams to ensure that it cannot be forced to provide an incorrect estimate by **Adversary**. This can be achieved by making the probability of $\mathcal{A}$'s failure on any specific selection of prefixes at most δ/t . Then, by the union bound, the probability that $\mathcal{A}$ errs on any of them is at most $t \cdot \delta/t = \delta$. Otherwise, all estimates throughout the stream are correct, independently of what **Adversary** does. Hence, a robust **Algorithm** in the τ -Stream Adversary Model can be achieved by using $\mathcal{A}$ and setting its error probability to at most δ/m^τ on any fixed input. This requires $M(\epsilon, \delta/m^\tau)$ space.

We can also express the space requirement as a function of $M(\epsilon, 1/10)$ by invoking Lemma 25. We run $\lceil 12 \ln(m^\tau/\delta) \rceil$ independent copies of $\mathcal{A}$, each erring with probability at most $1/10$ and using $M(\epsilon, 1/10)$ space. The median of their estimates is a correct solution to the (f, ϵ) -estimation problem in the oblivious setting with probability at least δ/m^τ . Due to our analysis, this a robust algorithm in τ -Stream Adversary Model and uses $O(\tau \log m + \log(1/\delta)) \cdot M(\epsilon, 1/10)$ space. ◀

6.3 Robust Algorithm for Bounded Memory Adversaries

We apply Lemma 26 to obtain a robust algorithm in the Bounded-Memory Adversary Model.

► **Theorem 27.** *Let $\mathcal{A}$ be an order-independent oblivious streaming algorithm that uses $M(\epsilon, \delta)$ space to provide a correct solution to an (f, ϵ) -estimation problem with probability $1 - \delta$ on streams of at most m updates for any $\epsilon \in (0, 1)$ and $\delta \in (0, 1)$. For any $\epsilon \in (0, 1)$ and any $\delta \in (0, 1/10)$, there is a robust streaming algorithm for the (f, ϵ) -estimation problem in the Bounded-Memory Adversary Model with $k \geq 0$ bits of persistent memory, that succeeds with probability at least $1 - \delta$ and uses $M(\epsilon/3, 1/10) \cdot O\left(2^k \cdot \frac{\log \alpha}{\epsilon} \cdot \log m + \log(1/\delta)\right)$ space, where f 's range is in $\{0\} \cup [1, \alpha]$ for $\alpha \geq 2$.*

Proof. First consider $\mathcal{A}$ with its parameters adjusted so that it solves the $(f, \epsilon/3)$ -estimation problem with probability at least $9/10$ on a stream of length at most m . When it succeeds, it produces an estimate y such that $y_* \leq y < (1 + \epsilon/3)y_*$, where y_* is the exact value of f on the final frequency vector. We modify $\mathcal{A}$ to round y to a small set of options before outputting it. In particular, Lemma 13 with its parameter ϵ set to $\epsilon/3$ gives us a small set $\mathcal{N}$

of values such that every correct estimate for the $(f, \epsilon/3)$ -estimation problem can be rounded up to a value y' in $\mathcal{N}_\star \stackrel{\text{def}}{=} \{0\} \cup \mathcal{N}$. The size of $\mathcal{N}$ – and therefore, also of $\mathcal{N}_\star$ – is of order $O(\epsilon^{-1} \log \alpha)$. More specifically, the rounding works as follows:

- If $y \in (-\infty, 1)$, we set $y' = 0$.
- If $y \in [1, \alpha]$, we round it up to the closest value in $\mathcal{N}$ that is greater or equal to it. Such a value exists due to the properties of $\mathcal{N}$, which was given by Lemma 13. This is the value to which we set y' in this case. Due to the properties of $\mathcal{N}$, we have $y \leq y' < (1 + \epsilon/3)y$.
- Finally, if $y > \alpha$, we treat it as α and handle as in the previous case (which actually means outputting $y' = \alpha$).

We now claim that if y is a correct estimate, by which we mean that it is a $(1 + \epsilon/3)$ -multiplicative approximation to $y_\star$, then y' is also a correct estimate, by which we mean that it is a $(1 + \epsilon)$ -multiplicative approximation to $y_\star$. To see this consider different scenarios. First, if $y \in (-\infty, 0) \cup (0, 1)$, y is not a correct estimate, so we can output anything, and we output $0 \in \mathcal{N}_\star$. Second if $y = 0$ and it is the correct estimate, $y_\star$ has to be 0 as well, and since we output $y' = 0$, we output the exact value of f . Next if $y \in [1, \alpha]$ and it is a correct estimate, we round it up to a value $y' \in \mathcal{N}$ such that $y \leq y' \leq (1 + \epsilon/3)y$. In this case, on the one hand, $y_\star \leq y \leq y'$, and on the other, $y' < (1 + \epsilon/3)y < (1 + \epsilon/3)(1 + \epsilon/3)y_\star < (1 + \epsilon)y_\star$, which means that y' that the algorithm outputs is a $(1 + \epsilon)$ -multiplicative approximation. Finally, if $y > \alpha$ and it is a correct estimate, then α is also a correct estimate, i.e., a $(1 + \epsilon/3)$ -multiplicative approximation to $y_\star$, because the range of possible values of f is bounded by α from above. In this case, we can use $y = \alpha$ instead and this case reduces to the previous one. In summary, y' is a $(1 + \epsilon)$ -multiplicative approximation to $y_\star$ with probability at least $9/10$ and it always belongs to set $\mathcal{N}_\star$.

It follows that the modified version of $\mathcal{A}$ can produce at most $|\mathcal{N}_\star| = O(\epsilon^{-1} \log \alpha)$ different estimates that can be passed to **Adversary**. Additionally, the persistent memory of **Adversary** can be in at most 2^k different states, which means there are at most $O(2^k \epsilon^{-1} \log \alpha)$ different input states for **Adversary** before randomness is considered. Hence, for a given input state, **Adversary** always generates an update from the same distribution. We can then reduce this scenario to the τ -Stream Adversary Model for $\tau = O(2^k \epsilon^{-1} \log \alpha)$ by observing that we could pre-generate a stream of **Adversary**'s updates in each possible input state. Then any possible sequence of **Adversary**'s updates is a result of merging these streams after the first initial update. Hence, Lemma 26 yields the desired result. ◀

6.4 Applications

Moment estimation. The most important application of our framework is to moment estimation. Note that for any fixed power $p \geq 0$, since there are at most m updates, the maximum value of the moment function is bounded by m^p , if $p \geq 1$, or m , if $p \in [0, 1]$. Best known turnstile streaming algorithms for moment estimation are based on linear sketching, which implies that they are order invariant and can be used in our framework. More specifically, two papers by Kane, Nelson, and Woodruff [20, 19] show that, for any fixed constant $p \in [0, 2]$, the (F_p, ϵ) -estimation problem can be solved via a linear sketching algorithm in $\tilde{O}(\epsilon^2 \log m)$ bits⁶ of space with probability $9/10$ in the oblivious streaming model. This leads to the following corollary of Theorem 27.

⁶ The $\tilde{O}$ notation used here hides polylogarithmic factors, i.e., $\tilde{O}(f(\epsilon, m))$ denotes $O(f(\epsilon, m) \log^c f(\epsilon, m))$ for some constant $c > 0$. This upper bound has been simplified using our assumption that all updates are unit updates and $n = \text{poly}(m)$.

► **Corollary 28.** *For any fixed $p \in [0, 2]$, the (F_p, ϵ) -estimation problem on streams with unit turnstile updates can be solved in the Bounded-Memory Adversary Model against an adversary with access to k bits of persistent memory by a robust algorithm that uses $\tilde{O}(2^k \epsilon^{-3} \log^3 m)$ bits of space with probability $9/10$.*

(Note that this result easily extends to integer updates from a range $\{-M, -M+1, \dots, M-1, M\}$, for any M . Updates from a range wider than $\{-1, 1\}$, which corresponds to unit updates, simply increase the range of possible values, i.e., α . More specifically, without going into all details, this comes at the cost of replacing at most two of the $\log m$ factors with $\log(Mn)$.)

Triangle counting. Another application concerns graph streaming. We focus on the fundamental triangle counting problem: here we are given a turnstile stream of edge updates to an unweighted graph on n nodes and must return an estimate to the number of triangles T . The stream can be viewed as updating a frequency vector of size $O(n^2)$ of edges. Trivially, $T = O(n^3)$. We use the classic algorithm of Bar-Yossef, Kumar, and Sivakumar [3] that solves this problem via a black-box reduction to F_0 , F_1 , and F_2 moment estimation of the aforementioned edge frequency vector. While it is not the state of the art classical streaming algorithm (see the work of Jayaram and Kallaugher [18] for a thorough discussion of the problem), it highlights the usefulness of Theorem 27.

► **Corollary 29.** *We write n to denote the number of vertices in the input graph. Consider the $(\text{triangles}, \epsilon)$ -estimation problem in which $\text{triangles}(G)$ is the number of triangles in the input graph G , represented here as a frequency vector of edges of length $\binom{n}{2}$.*

This triangle estimation problem on turnstile streams of length at most $\text{poly}(n)$ can be solved in the Bounded-Memory Adversary Model against an adversary who has k bits of persistent space by a robust algorithm that uses $\tilde{O}(2^k \epsilon^{-3} (n^3/T)^2)$ space and succeeds with probability $9/10$, where T is the number of triangles.

7 Discussion and Open Problems

We conclude the paper with a few natural open problems stemming from our work. The most important question in our opinion is whether one can avoid the exponential dependence on k , the size of the persistent memory. We restate the main question of the introduction:

► **Question 30.** *Can we avoid the $2^{\Omega(k)}$ dependence in the size k of the persistent memory in our results for deterministic adversaries in Corollary 15 and for order-independent algorithms in Theorem 27? In particular, can we obtain dependence polynomial in k ?*

At a high level, we incur this dependence because we need to handle all 2^k many possible different states of the persistent memory. Progress towards our main open question could also have downstream implications in the original robust streaming setting of Ben-Eliezer et al. [5] (but now with *deletions*!), since a large persistent memory allows the adversary to remember more of the interaction with the streaming algorithm and correlate future inputs with many past estimates, thus approaching the power of the original adversary of [5].

A related question concerns the τ -Stream Adversary Model (Definition 24), in which the adversary has unlimited persistent memory, but must choose each input from one of τ streams generated in advance. We ask if limiting the adversary's persistent memory in this case, analogous to our main model, the Bounded-Memory Adversary Model, can result in more space-efficient streaming algorithms.

► **Question 31.** *Can we improve the dependence on τ in Lemma 26 if we assume that the adversary has bounded persistent memory? Is dependence sublinear in τ possible?*

Note that progress towards the above question would directly improve Theorem 27 and Question 30, since the proof of Theorem 27 proceeds by reducing the randomized adversary (in the Bounded-Memory Adversary Model) to one which interleaves between exponentially many (in the size of the persistent memory) different streams.

We now ask about improving the dependence on α , the range of f , in our theorem statements. This is especially important for settings such as heavy-hitter detection, in which valid outputs include many different subsets of the domain $[n]$. Note that in the classic streaming setting, it is possible to obtain space bounds depending on the number of heavy hitters [9], so we ask an analogous question for our Bounded-Memory Adversary Model.

► **Question 32.** *Can our results be extended to the case where the number of different valid outputs is large (such as the range of f being exponential in n)? In particular, what can we say about the problem of identifying heavy hitters in the Bounded-Memory Adversary Model?*

In this case, we can think of α as being exponential in n . Our current bounds do not apply here in a satisfactory way. For example, we can set $\alpha = 2^n$ in our theorem statements, so our space bound reduces to linear in n , which can trivially be satisfied by explicitly keeping track of all coordinates. At the heart of it, the challenge comes from the fact that we cannot efficiently perform “output rounding” for heavy hitters as we did in, for instance, Theorem 14. In particular, it is easy to construct an adversary, even with very limited persistent memory, that can cycle through many different subsets of answers. For example, suppose **Adversary** gives $(1, 1)$ as the first update. Then **Algorithm** must output $\{1\}$. Then **Adversary** gives $(2, 1)$, so the algorithm’s output must be $\{1, 2\}$. Then **Adversary** gives $(1, -1)$, so the right set of heavy hitters is $\{2\}$, and so on. In other words, by looking at the very last output, **Adversary** can determine which elements have a frequency of one and then correspondingly add or delete elements to update the set of heavy hitters.

The next question concerns our lower bound construction of Theorem 16. The result and proof apply specifically to the case of $f = F_2$, i.e., the goal is to compute the second moment of the stream. It remains open whether a similar attack could be deployed against other important functions, such as F_0 (distinct elements).

► **Question 33.** *Is it possible for an adversary in the Bounded-Memory Adversary Model to enforce a stream with both high density and high flip number for F_0 ? In particular, is it possible for an adversary with a persistent memory of size $o(\log m)$ to make these quantities $m^{\Omega(1)}$? Conversely, what if we give the adversary $k = O(\log m)$ bits of persistent memory?*

We suspect the answer is negative, since one item can only influence the number of distinct elements by 1. Thus intuitively, if the adversary is exactly keeping track of such polynomially many elements exactly, then it requires polynomial space. However, the adversary itself could be simulating a distinct elements algorithm, using its persistent memory, in which case $O(\log m)$ space may suffice. We leave this as an intriguing open question.

Next we turn to Theorem 14. In its proof, we explicitly kept track of the frequency vector, which was forced to be sparse due to the rounding performed by the streaming algorithm. One could imagine making the algorithm even more space efficient by approximately tracking the sparse frequency vector, but this will likely depend on the underlying properties of f which we abstracted away.

► **Question 34.** *Is the space bound of Theorem 14 the right bound? Can we classify a broad family of functions f that use even smaller space, e.g., by approximately tracking the frequency vector?*

Lastly, we ask if the techniques of Woodruff and Zhou [30] can deal with bounded memory adversaries for the specific problem of F_2 estimation.

► **Question 35.** *What are the limits of the techniques of Woodruff and Zhou [30] for estimating F_2 in the Bounded-Memory Adversary Model?*

Our construction of a stream with high flip number and high density heavily relies on the use of a single heavy hitter. Toward answering the above question, it would be interesting to understand whether this is inherent or just an artifact of the specific construction. If it is indeed an inherent feature of the memoryless adversarial setting, then this could mean that dense-sparse strategies such as those of Woodruff and Zhou [30], which leverage deterministic algorithms for heavy-hitter detection, could also be useful for achieving efficient algorithms that are robust against memoryless adversaries.

We wish to conclude the discussion by pointing out the problem of *missing item finding* (MIF), which has been explored by Stoeckl and Chakrabarti [29, 8]. In this problem, the input is a stream of elements from universe $[n]$, and at each point throughout a stream of length $m < n$, the algorithm is asked to output an element that has not appeared in the stream so far. Interestingly, MIF is a problem in which memoryless deterministic adversaries do turn out to have a significant advantage over oblivious adversaries in some regimes. For instance, the “echo” strategy in which the adversary uses the last output of the algorithm as the next stream element turns out very effective, because it can be used to force the algorithm to find a new missing element. However, MIF is not a problem for which multiplicative approximation has any meaningful interpretation. Since many incompatible outputs are possible in this problem, and on each, the adversary can act differently, this seems to fall into the topic considered in Question 32. As such, the results we obtain here are incompatible with the existing literature on MIF, but MIF could be a good starting point for further exploration of the power of memoryless and low-memory adversaries in various types of streaming problems.

References

- 1 David J. Aldous and James Allen Fill. Reversible markov chains and random walks on graphs, 2002. Unfinished monograph; recompiled 2014. Online at <http://www.stat.berkeley.edu/~aldous/RWG/book.html>.
- 2 Idan Attias, Edith Cohen, Moshe Shechner, and Uri Stemmer. A framework for adversarial streaming via differential privacy and difference estimators. *Algorithmica*, 86(11):3339–3394, 2024. doi:10.1007/S00453-024-01259-8.
- 3 Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545464>.
- 4 Omri Ben-Eliezer, Talya Eden, and Krzysztof Onak. Adversarially robust streaming via dense-sparse trade-offs. In *5th Symposium on Simplicity in Algorithms (SOSA@SODA)*, pages 214–227, 2022. doi:10.1137/1.9781611977066.15.
- 5 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. *J. ACM*, 69(2):17:1–17:33, 2022. doi:10.1145/3498334.

- 6 Omri Ben-Eliezer and Eylon Yogev. The adversarial robustness of sampling. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 49–62, 2020. doi:10.1145/3375395.3387643.
- 7 Vladimir Braverman, Avinatan Hassidim, Yossi Matias, Mariano Schain, Sandeep Silwal, and Samson Zhou. Adversarial robustness of streaming algorithms through importance sampling. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:3544–3557, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/1d01bd2e16f57892f0954902899f0692-Abstract.html>.
- 8 Amit Chakrabarti and Manuel Stoeckl. Finding missing items requires strong forms of randomness. In *39th Computational Complexity Conference (CCC)*, pages 28:1–28:20, 2024. doi:10.4230/LIPIcs.CCC.2024.28.
- 9 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *29th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 693–703, 2002. doi:10.1007/3-540-45465-9_59.
- 10 Yeshwanth Cherapanamjeri, Sandeep Silwal, David Woodruff, Fred Zhang, Qiuyi Zhang, and Samson Zhou. Robust algorithms on adaptive inputs from bounded adversaries. In *11th International Conference on Learning Representations (ICLR)*, 2023. URL: <https://openreview.net/forum?id=I29KtORwChs>.
- 11 Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, Moshe Shechner, and Uri Stemmer. On the robustness of counts sketch to adaptive inputs. In *International Conference on Machine Learning (ICML)*, pages 4112–4140. PMLR, 2022. URL: <https://proceedings.mlr.press/v162/cohen22a.html>.
- 12 Edith Cohen, Jelani Nelson, Tamás Sarlós, and Uri Stemmer. Tricking the hashing trick: A tight lower bound on the robustness of CountSketch to adaptive inputs. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*, pages 7235–7243, 2023. doi:10.1609/AAAI.V37I6.25882.
- 13 Elena Gribelyuk, Honghao Lin, David P. Woodruff, Huacheng Yu, and Samson Zhou. A strong separation for adversarially robust ℓ_0 estimation for linear sketches. In *65th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2318–2343, 2024.
- 14 Elena Gribelyuk, Honghao Lin, David P. Woodruff, Huacheng Yu, and Samson Zhou. Lifting linear sketches: Optimal bounds and adversarial robustness. In *57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 395–406, 2025. doi:10.1145/3717823.3718227.
- 15 Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 121–130, 2013. doi:10.1145/2488608.2488624.
- 16 Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. *J. ACM*, 69(6):42:1–42:14, 2022. doi:10.1145/3556972.
- 17 Rajesh Jayaram. *Sketching and Sampling Algorithms for High-Dimensional Data*. PhD thesis, Carnegie Mellon University, 2021.
- 18 Rajesh Jayaram and John Kallaugher. An optimal algorithm for triangle counting in the stream. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, pages 11:1–11:11, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.11.
- 19 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1178, 2010. doi:10.1137/1.9781611973075.93.
- 20 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2010. doi:10.1145/1807085.1807094.

- 21 Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. Separating adaptive streaming from oblivious streaming using the bounded storage model. In *Advances in Cryptology – CRYPTO 2021*, pages 94–121, 2021. doi:10.1007/978-3-030-84252-9_4.
- 22 Yi Li, Huy L Nguyen, and David P Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *46th Annual ACM symposium on Theory of Computing (STOC)*, pages 174–183, 2014. doi:10.1145/2591796.2591812.
- 23 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.
- 24 Boaz Menuhin and Moni Naor. Keep That Card in Mind: Card Guessing with Limited Memory. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, pages 107:1–107:28, 2022. doi:10.4230/LIPIcs.ITCS.2022.107.
- 25 Boaz Menuhin and Moni Naor. Shuffling cards when you are of very little brain: Low memory generation of permutations. *arXiv preprint 2505.01287*, 2025. To appear in FOCS 2025.
- 26 S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers, 2005.
- 27 Dragos-Florian Ristache and Krzysztof Onak. The adaptive use of count-min sketch: What is safe and what is not? In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD)*, pages 2432–2443, 2025.
- 28 Menachem Sadigurschi, Moshe Shechner, and Uri Stemmer. Relaxed Models for Adversarial Streaming: The Bounded Interruptions Model and the Advice Model. In *31st Annual European Symposium on Algorithms (ESA)*, pages 91:1–91:14, 2023. doi:10.4230/LIPIcs.ESA.2023.91.
- 29 Manuel Stoeckl. Streaming algorithms for the missing item finding problem. In *34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 793–818, 2023. doi:10.1137/1.9781611977554.CH32.
- 30 David Woodruff and Samson Zhou. Adversarially robust dense-sparse tradeoffs via heavy-hitters. In *38th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024. URL: <https://openreview.net/forum?id=MPidsCd9e7>.
- 31 David P. Woodruff and Samson Zhou. Tight Bounds for Adversarially Robust Streams and Sliding Windows via Difference Estimators. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1183–1196, 2022.