A Framework for Adversarial Streaming via Differential Privacy and Difference Estimators

Idan Attias
Ben-Gurion University, Beer Sheva, Israel

Edith Cohen
Google Research, Mountain View, CA, USA
Tel Aviv University, Israel

Moshe Shechner
Tel Aviv University, Israel

Uri Stemmer
Tel Aviv University, Israel
Google Research, Herzliya, Israel

Abstract

Classical streaming algorithms operate under the (not always reasonable) assumption that the input stream is fixed in advance. Recently, there is a growing interest in designing robust streaming algorithms that provide provable guarantees even when the input stream is chosen adaptively as the execution progresses. We propose a new framework for robust streaming that combines techniques from two recently suggested frameworks by Hassidim et al. [NeurIPS 2020] and by Woodruff and Zhou [FOCS 2021]. These recently suggested frameworks rely on very different ideas, each with its own strengths and weaknesses. We combine these two frameworks into a single hybrid framework that obtains the “best of both worlds”, thereby solving a question left open by Woodruff and Zhou.

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

Keywords and phrases Streaming, adversarial robustness, differential privacy

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.8

Related Version Full Version: https://arxiv.org/abs/2107.14527

Funding Idan Attias: Partially supported by the Vatat Scholarship from the Israeli Council for Higher Education, Kreitman School of Advanced Graduate Studies, and the Lynn and William Frankel Center for Computer Science at Ben-Gurion University.
Edith Cohen: Partially supported by the Israel Science Foundation (grant 1505/19).
Moshe Shechner: Partially supported by the Israel Science Foundation (grant 1871/19).
Uri Stemmer: Partially supported by the Israel Science Foundation (grant 1871/19) and by Len Blavatnik and the Blavatnik Family foundation.

1 Introduction

Streaming algorithms are algorithms for processing large data streams while using only a limited amount of memory, significantly smaller than what is needed to store the entire data stream. Data streams occur in many applications including computer networking, databases, and natural language processing. The seminal work of Alon, Matias, and Szegedy [3] initiated an extensive theoretical study and further applications of streaming algorithms.

* See [4] for a full version of this work.
In this work we focus on streaming algorithms that aim to maintain, at any point in time, an approximation for the value of some (predefined) real-valued function of the input stream. Such streaming algorithms are sometimes referred to as strong trackers. For example, this predefined function might count the number of distinct elements in the stream. Formally,

**Definition 1.1.** Let $A$ be an algorithm that, for $m$ rounds, obtains an element from a domain $X$ and outputs a real number. Algorithm $A$ is said to be a strong tracker for a function $F : X^* \rightarrow \mathbb{R}$ with accuracy $\alpha$, failure probability $\delta$, and stream length $m$ if the following holds for every sequence $\vec{u} = (u_1, \ldots, u_m) \in X^m$. Consider an execution of $A$ on the input stream $\vec{u}$, and denote the answers given by $A$ as $\vec{z} = (z_1, \ldots, z_m)$. Then,

$$\Pr[\forall i \in [m] : z_i \in (1 \pm \alpha) \cdot F(u_1, \ldots, u_i)] \geq 1 - \delta,$$

where the probability is taken over the coins of algorithm $A$.

While Definition 1.1 is certainly not the only possible definition of streaming algorithms, it is rather standard. Note that in this definition we assume that the input stream $\vec{u}$ is fixed in advance. In particular, we assume that the choice for the elements in the stream is independent from the internal randomness of $A$. This assumption is crucial for the analysis (and correctness) of many of the existing streaming algorithms. We refer to algorithms that utilize this assumption as oblivious streaming algorithms. In this work we are interested in the setting where this assumption does not hold, often called the adversarial setting.

### 1.1 The adversarial streaming model

The adversarial streaming model, in various forms, was considered by [1, 2, 7–11, 14, 15, 18, 19, 21, 23]. We give here the formulation presented by Ben-Eliezer et al. [7]. The adversarial setting is modeled by a two-player game between a (randomized) StreamingAlgorithm and an Adversary. At the beginning, we fix a function $F : X^* \rightarrow \mathbb{R}$. Then the game proceeds in rounds, where in the $i$th round:

1. The **Adversary** chooses an update $u_i \in X$ for the stream, which can depend, in particular, on all previous stream updates and outputs of StreamingAlgorithm.
2. The **StreamingAlgorithm** processes the new update $u_i$ and outputs its current response $z_i \in \mathbb{R}$.

The goal of the **Adversary** is to make the **StreamingAlgorithm** output an incorrect response $z_i$ at some point $i$ in the stream. For example, in the distinct elements problem, the adversary’s goal is that at some step $i$, the estimate $z_i$ will fail to be a $(1 + \alpha)$-approximation of the true current number of distinct elements.

In this work we present a new framework for transforming an oblivious streaming algorithm into an adversarially-robust streaming algorithm. Before presenting our framework, we first elaborate on the existing literature and the currently available frameworks.

### 1.2 Existing framework: Ben-Eliezer et al. [7]

To illustrate the results of [7], let us consider the distinct elements problem, in which the function $F$ counts the number of distinct elements in the stream. Observe that, assuming that there are no deletions in the stream, this quantity is monotonically increasing. Furthermore, since we are aiming for a multiplicative error, the number of times we need to modify the estimate we release is quite small (it depends logarithmically on the stream length $m$). Informally, the idea of [7] is to run several independent copies of an oblivious algorithm (in
parallel), and to use each copy to release answers over a part of the stream during which the estimate remains constant. In more detail, the generic transformation of [7] (applicable not only to the distinct elements problem) is based on the following definition.

**Definition 1.2** (Flip number [7]). Given a function $F$, the $(\alpha, m)$-flip number of $F$, denoted as $\lambda_{\alpha, m}(F)$, or simply $\lambda$ in short, is the maximal number of times that the value of $F$ can change (increase or decrease) by a factor of $(1 + \alpha)$ during a stream of length $m$.

**Remark 1.3.** In the technical sections of this work, we sometimes refer to the flip number of the given stream (w.r.t. the target function), which is a more fine-tuned quantity.

**Example 1.4.** Assuming that there are no deletions in the stream (a.k.a. the insertion only model), the $(\alpha, m)$-flip number of the distinct elements problem is at most $O\left(\frac{1}{\alpha} \log m\right)$. However, if deletions are allowed (a.k.a. the turnstile model), then the flip number of this problem could be as big as $\Omega(m)$.

The generic construction of [7] for a function $F$ is as follows.

1. Instantiate $\lambda$ independent copies of an oblivious streaming algorithm for the function $F$, and set $j = 1$.
2. When the next update $u_i$ arrives:
   a. Feed $u_i$ to all of the $\lambda$ copies.
   b. Release an estimate using the $j$th copy (rounded to the nearest power of $(1 + \alpha)$). If this estimate is different than the previous estimate, then set $j \leftarrow j + 1$.

Ben-Eliezer et al. [7] showed that this can be used to transform an oblivious streaming algorithm for $F$ into an adversarially robust streaming algorithm for $F$. In addition, the overhead in terms of memory is only $\lambda$, which is small in many interesting settings.

The simple, but powerful, observation of Ben-Eliezer et al. [7], is that by “using every copy at most once” we can break the dependencies between the internal randomness of our algorithm and the choice for the elements in the stream. Intuitively, this holds because the answer is always computed using a “fresh copy” whose randomness is independent from the choice of stream items.

### 1.3 Existing framework: Hassidim et al. [19]

Hassidim et al. [19] showed that, in fact, we can use every copy of the oblivious algorithm much more than once. In more detail, the idea of Hassidim et al. is to protect the internal randomness of each of the copies of the oblivious streaming algorithm using differential privacy [13]. Hassidim et al. showed that this still suffices in order to break the dependencies between the internal randomness of our algorithm and the choice for the elements in the stream. This resulted in an improved framework where the space blowup is only $\sqrt{\lambda}$ (instead of $\lambda$). Informally, the framework of [19] is as follows.

1. Instantiate $\sqrt{\lambda}$ independent copies of an oblivious streaming algorithm for the function $F$.
2. When the next update $u_i$ arrives:
   a. Feed $u_i$ to all of the $\sqrt{\lambda}$ copies.
   b. Aggregate all of the estimates given by the $\sqrt{\lambda}$ copies, and compare the aggregated estimate to the previous estimate. If the estimate had changed “significantly”, output the new estimate. Otherwise output the previous output.
In order to efficiently aggregate the estimates in Step 2b, this framework crucially relied on the fact that all of the copies of the oblivious algorithm are “the same” in the sense that they compute (or estimate) exactly the same function of the stream. This allowed Hassidim et al. to efficiently aggregate the returned estimates using standard tools from the literature on differential privacy. The intuition is that differential privacy allows us to identify global properties of the data, and hence, aggregating several numbers (the outcomes of the different oblivious algorithms) is easy if they are very similar.

1.4 Existing framework: Woodruff and Zhou [29]

Woodruff and Zhou [29] presented an adversarial streaming framework that builds on the framework of Ben-Eliezer et al. [7]. The new idea of [29] is that, in many interesting cases, the oblivious algorithms we execute can be modified to track different (but related) functions, that require less space while still allowing us to use (or combine) several of them at any point in time in order to estimate $F$.

To illustrate this, consider a part of the input stream, say from time $t_1$ to time $t_2$, during which the target function $F$ doubles its value and is monotonically increasing. More specifically, suppose that we already know (or have a good estimation for) the value of $F$ at time $t_1$, and we want to track the value of $F$ from time $t_1$ till $t_2$. Recall that in the framework of [7] we only modify our output once the value of the function has changed by more than a $(1 + \alpha)$ factor. As $F(t_2) \leq 2 \cdot F(t_1)$, we get that between time $t_1$ and $t_2$ there are roughly $1/\alpha$ time points at which we need to modify our output. In the framework of [7], we need a fresh copy of the oblivious algorithm for each of these $1/\alpha$ time points. For concreteness, let us assume that every copy uses space $1/\alpha^2$ (which is the case if, e.g., $F = F_2$), and hence the framework of [7] requires space $1/\alpha^3$ to track the value of the function $F$ from $t_1$ till $t_2$.

In the framework of [29], on the other hand, this will cost only $1/\alpha^2$. We now elaborate on this improvement. As we said, from time $t_1$ till $t_2$ there are $1/\alpha$ time points on which we need to modify our output. Let us denote these time points as $t_1 = w_0 < w_1 < w_2 < \cdots < w_{1/\alpha} = t_2$. In the framework of [29], the oblivious algorithms we execute are tracking differences between the values of $F$ at specific times, rather than tracking the value of $F$ directly. (These algorithms are called difference estimators, or DE in short.) In more detail, suppose that for every $j \in \{0, 1, 2, 3, \ldots, \log \frac{1}{\alpha}\}$ and every $i \in \{2^j, 2 \cdot 2^j, 3 \cdot 2^j, 4 \cdot 2^j, \ldots, \frac{1}{\alpha}\}$ we have an oblivious algorithm (a difference estimator) for estimating the value of $|F(w_i) - F(w_{i-\alpha})|$. We refer to the index $j$ as the level of the oblivious algorithm. So there are $\log \frac{1}{\alpha}$ different levels, where we have a different number of oblivious algorithms for each level. (For level $j = 0$ we have $1/\alpha$ oblivious algorithms and for level $j = \log \frac{1}{\alpha}$ we have only a single oblivious algorithm.)

Note that given all of these oblivious algorithms, we could compute an estimation for the value of the target function $F$ at each of the time points $w_1, \ldots, w_{1/\alpha}$ (and hence for every time $t_1 \leq t \leq t_2$) by summing the estimations of (at most) one oblivious algorithm from each level. For example, an estimation for the value of $F(w_{\frac{1}{\alpha}^{-1} + 1})$ can be obtained by combining estimations as follows:

$$F\left(w_{\frac{1}{\alpha}^{-1} + 1}\right) = F\left(w_0\right) + \left[ F\left(w_{\frac{1}{\alpha}}\right) - F\left(w_0\right)\right] + \left[ F\left(w_{\frac{1}{\alpha}^{-1}}\right) - F\left(w_{\frac{1}{\alpha}}\right)\right] + \left[ F\left(w_{\frac{1}{\alpha}^{-1} + 1}\right) - F\left(w_{\frac{1}{\alpha}^{-1}}\right)\right].$$

1 Note that these time points are not known to the algorithm in advance. Rather, the algorithm needs to discover them “on the fly”. To simplify the presentation, in Section 1.4 we assume that these time points are known in advance.

2 Specifically, in order to reach the estimated value of $F$ at time $w_t$ one can add the estimations of difference estimators of levels corresponds to the binary representation of $t$. That is, at most one of each level $j$. 
As we sum at most \( \log \frac{1}{\alpha} \) estimations, this decomposition increases our estimation error only by a factor of \( \log \frac{1}{\alpha} \), which is acceptable. The key observation of [29] is that the space complexity needed for an oblivious algorithm at level \( j \) decreases when \( j \) decreases (intuitively because in lower levels we need to track smaller differences, which is easier). So, even though in level \( j=10 \) we have more oblivious algorithms than in level \( 20 \), these oblivious algorithms are cheaper than in level \( 20 \) such that the overall space requirements for levels \( j=10 \) and level \( j=20 \) (or any other level) is the same. Specifically, [29] showed that (for many problems of interest, e.g., for \( F_2 \)) the space requirement of a difference estimator at level \( j \) is \( O(2^j / \alpha) \). We run \( O(2^{-j} / \alpha) \) oblivious algorithms for level \( j \), and hence, the space needed for level \( j \) is \( O(2^{-j} / \alpha \cdot 2^j / \alpha) = O(1/\alpha^2) \). As we have \( \log(1/\alpha) \) levels, the overall space we need to track the value of \( F \) from time \( t_1 \) till \( t_2 \) is \( \tilde{O}(1/\alpha^2) \). This should be contrasted with the space required by [7] for this time segment, which is \( O(1/\alpha^3) \).

1.5 Our results

The framework of [29] is very effective for the insertion-only model. However, there are two challenges that need to be addressed in the turnstile setting: (1) We are not aware of non-trivial constructions for difference estimators in the turnstile setting, and hence, the framework of [29] is not directly applicable to the turnstile setting.\(^3\) (2) Even assuming the existence of a non-trivial difference estimator, the framework of [29] obtains sub-optimal results in the turnstile setting.

To overcome the first challenge, we introduce a new monitoring technique, that aims to identify time steps at which we cannot guarantee correctness of our difference estimators (in the turnstile setting), and reset the system at these time steps. This will depend on the specific application at hand (the target function) and hence, we defer the discussion on our monitoring technique to Section 5 where we discuss applications of our framework.

We now focus on the second challenge (after assuming the existence of non-trivial difference estimators). To illustrate the sub-optimality of the framework of [29], let us consider a simplified turnstile setting in which the input stream can be partitioned into \( k \) time segments during each of which the target function is monotonic, and increases (or decreases) by at most a factor of 2 (or 1/2). Note that \( k \) can be very large in the turnstile model (up to \( O(m) \)).

With the framework of [29], we would need space \( \tilde{O} \left( \frac{m}{\alpha} \right) \) to track the value of \( F_2 \) throughout such an input stream. The reason is that, like in the framework of [7], the robustness guarantees are achieved by making sure that every oblivious algorithm is “used only once”. This means that we cannot reuse the oblivious algorithms across the different segments, and hence, the space complexity of [29] scales linearly with the number of segments \( k \).

To mitigate this issue, we propose a new construction that combines the frameworks of [29] with the framework of [19]. Intuitively, in our simplified example with the \( k \) segments, we want to reuse the oblivious algorithms across different segments, and protect their internal randomness with differential privacy to ensure robustness. However, there is an issue here. Recall that the framework of [19] crucially relied on the fact that all of the copies of the oblivious algorithm are “the same” in the sense that they compute the same function exactly. This allowed [19] to efficiently aggregate the estimates in a differentially private manner. However, in the framework of [29], the oblivious algorithms we maintain are fundamentally different from each other, tracking different functions. Specifically, every difference estimator is tracking the value of \( |F(t) - F(e)| \) for a unique enabling time \( e < t \) (where \( t \) denotes the

\(^3\) Moshe Shechner and Samson Zhou. Personal communication, 2022.
current time). That is, every difference estimator necessarily has a different enabling time, and hence, they are not tracking the same function, and it is not clear how to aggregate their outcomes with differential privacy.

**Toggle Difference Estimator (TDE).** To overcome the above challenge, we present an extension to the notion of a difference estimator, which we call a *Toggle Difference Estimator* (see Definition 2.3). Informally, a toggle difference estimator is a difference estimator that allows us to modify its *enabling time* on the go. This means that a TDE can track, e.g., the value of \( F(t) - F(e_1) \) for some (previously given) enabling time \( e_1 \), and then, at some later point in time, we can instruct the same TDE to track instead the value of \( F(t) - F(e_2) \) for some other enabling time \( e_2 \). We show that this extra requirement from the difference estimator comes at a very low cost in terms of memory and runtime. Specifically, in Section 4 we present a generic (efficiency preserving) method for generating a TDE from a DE.

Let us return to our example with the \( k \) segments. Instead of using every oblivious algorithm only once, we reuse them across the different segments, where during any single segment all the TDE’s are instructed to track the appropriate differences that are needed for the current segment. This means that during every segment we have many copies of the “same” oblivious algorithm. More specifically, for every different *level* (as we explained above) we have many copies of an oblivious algorithm for that level, which is (currently) tracking the difference that we need. This allows our space complexity to scale with \( \sqrt{k} \) instead of linearly with \( k \) as the framework of [29]. To summarize this discussion, our new notion of TDE allows us to gain both the space saving achieved by differential privacy (as in the framework of [19]) and the space saving achieved by tracking the target function via differences (as in the framework of [29]).

**Remark 1.5.** The presentation given above (w.r.t. our example with the \( k \) segments) is oversimplified. Clearly, in general, we have no guarantees that an input (turnstile) stream can be partitioned into \( k \) such segments. This means that in the actual construction we need to calibrate our TDE’s across time segments in which the value of the target function is not monotone. See Section 3.1 for a more detailed overview of our construction and the additional modifications we had to introduce.

We are now ready to state our main result (for the formal statement see Theorem A.10). We present a framework for adversarial streaming for turnstile streams with bounded flip number \( \lambda \), for any function \( F \) for which the following algorithms exist:

1. An \( \alpha \)-accurate oblivious streaming algorithm \( E \) with space complexity \( \text{Space}(E) \).
2. An oblivious TDE streaming algorithm \( E_{\text{TDE}} \) (satisfying some conditions).

Under these conditions, our framework results in an \( O(\alpha) \)-accurate adversarially-robust algorithm with space\(^4 \) \( \tilde{O}\left(\sqrt{\alpha \cdot \lambda \cdot \text{Space}(E)}\right) \). In contrast, under the same conditions, the framework of [29] requires space \( \tilde{O}(\alpha \cdot \lambda \cdot \text{Space}(E)) \).

As we mentioned, we are not aware of non-trivial constructions for difference estimators that work in the turnstile setting. Nevertheless, in Section 5 we show that our framework is applicable to the problem of estimating \( F_2 \) (the second moment of the stream). To this end, we introduce the following notion that allows us to control the number of times we need to reset our system (which happens when we cannot guarantee correctness of our difference estimators).

\(^4\) Here \( \tilde{O} \) stands for omitting poly-logarithmic factors of \( \lambda, \alpha^{-1}, \delta^{-1}, n, m \).
\textbf{Definition 1.6 (Twist number).} The \((\alpha, m)\)-twist number of a stream \(S\) w.r.t. a functionality \(F\), denoted as \(\mu_{\alpha,m}(S)\), is the maximal \(\mu \in \{m\}\) such that \(S\) can be partitioned into \(2\mu\) disjoint segments \(S = P_0 \circ V_0 \circ \cdots \circ P_{\mu-1} \circ V_{\mu-1}\) (where \(\{P_i\}_{i \in [\mu]}\) may be empty) s.t. for every \(i \in [\mu]::

1. \(F(V_i) > \alpha \cdot F(P_0 \circ V_0 \circ \cdots \circ V_{i-1} \circ P_i)\)
2. \(|F(P_0 \circ V_0 \circ \cdots \circ P_i \circ V_i) - F(P_0 \circ V_0 \circ \cdots \circ P_i)| \leq \alpha \cdot F(P_0 \circ V_0 \circ \cdots \circ P_i)\)

In words, a stream has twist number \(\mu\) if there are \(\mu\) disjoint segments \(V_0, \ldots, V_{\mu-1} \subseteq S\) such that the value of the function on each of these segments is large (Condition 1), but still these segments do not change the value of the function on the entire stream by too much (Condition 2). Intuitively, the twist number bounds the number of regions in which a local view of the stream would suggest a large multiplicative change, but a global view would not. In Section 5 we leverage this notation and present the following result for \(F_2\) estimation.

\textbf{Theorem 1.7 (\(F_2\) Robust estimation, informal).} There exists an adversarially robust \(F_2\) estimation algorithm for turnstile streams of length \(m\) with a bounded \((O(\alpha), m)\)-flip number \(\lambda\) and a bounded \((O(\alpha), m)\)-twist number \(\mu\) that guarantees \(\alpha\)-accuracy (w.h.p.) using space complexity \(\tilde{O}\left(\frac{\sqrt{\lambda \mu}}{\alpha^2}\right)\).

This should be contrasted with the result of [19], who obtain space complexity \(\tilde{O}\left(\frac{\sqrt{\lambda}}{\alpha}\right)\) for robust \(F_2\) estimation in the turnstile setting. Hence, our new result is better whenever \(\mu \ll \lambda\).

\textbf{Example 1.8.} For \(F_2\) estimation in insertion-only streams, it holds that \(\mu = 0\) even though \(\lambda\) can be large. This is the case because, in insertion only streams, Conditions 1 and 2 from Definition 1.6 cannot hold simultaneously. Specifically, denote \(p = P_0 \circ \cdots \circ P_i\) and \(v = V_i\), and suppose that Condition 2 holds, i.e., \(\|p \circ v\|^2 - \|p\|^2 \leq \alpha \cdot \|p\|^2\). Hence, in order to show that Condition 1 does not hold, it suffices to show that \(\|v\|^2 \leq \|p \circ v\|^2 - \|p\|^2\), i.e., show that \(\|v\|^2 + \|p\|^2 \leq \|p \circ v\|^2\), i.e., show that \(v_1^2 + p_1^2 + \cdots + v_n^2 + p_n^2 \leq (v_1 + p_1)^2 + \cdots + (v_n + p_n)^2\), which trivially holds whenever \(p_i, v_i \geq 0\).

1.6 Other related works

Related to our work is the line of work on adaptive data analysis, aimed at designing tools for guaranteeing statistical validity in cases where the data is being accessed adaptively [5,12,17,20,22,24–28]. Recall that the difficulty in the adversarial streaming model arises from potential dependencies between the inputs of the algorithm and its internal randomness. As we mentioned, our construction builds on a technique introduced by [19] for using differential privacy to protect not the input data, but rather the internal randomness of algorithm. Following [19], this technique was also used by [6,16] for designing robust algorithms in other settings.

2 Preliminaries

In this work we consider input streams which are represented as a sequence of \textit{updates}, where every \textit{update} is a tuple containing an element (from a finite domain) and its (integer) weight. Formally,
Definition 2.1 (Turnstile stream). A stream of length \( m \) over a domain \([n]\) consists of a sequence of updates \((s_0, \Delta_0), \ldots, (s_{m-1}, \Delta_{m-1})\) where \( s_i \in [n] \) and \( \Delta_i \in \mathbb{Z} \). Given a stream \( S \in ([n] \times \mathbb{Z})^m \) and integers \( 0 \leq t_1 \leq t_2 \leq m - 1 \), we write \( S_{t_0}^{t_1} = ((s_{t_1}, \Delta_{t_1}), \ldots, (s_{t_2}, \Delta_{t_2})) \) to denote the sequence of updates from time \( t_1 \) till \( t_2 \). We also use the abbreviation \( S_t = S_t^t \) to denote the first \( t \) updates.

Let \( F : ([n] \times \mathbb{Z})^* \rightarrow \mathbb{R} \) be a function (for example \( F \) might count the number of distinct elements in the stream). At every time step \( t \), after obtaining the next element in the stream \((s_t, \Delta_t)\), our goal is to output an approximation for \( F(S_t) \). To simplify presentation we also denote \( F(t) = F(S_t) \) for \( t \in [m] \). We assume throughout the paper that \( \log(m) = \Theta(\log(n)) \) and that \( F \) is bounded polynomially in \( n \).

In Section 1, for the purpose of presentation, it was useful to refer to the quantity a flip number of a function. Our results are stated w.r.t a more refined quantity: a flip number of a stream.

Definition 2.2 (Flip number of a stream [7]). Given a function \( F \) and a stream \( S \) of length \( m \), the \((\alpha, m)\)-flip number of \( S \), denoted as \( \lambda_{\alpha}(S) \), is the maximal number of times that the value of \( F \) can change (increase or decrease) by a factor of \((1 + \alpha)\) during the stream \( S \).

Toggle Difference Estimator. For the purpose of our framework, we present an extension to the notion of a difference estimator (DE) from [29], which we call a toggle difference estimator (TDE). A difference estimator for a function \( F \) is an oblivious streaming algorithm, defined informally as follows. The difference estimator is initiated on time \( t = 1 \) and has a dynamically defined enabling time \( 1 \leq e \leq m \). Once that enabling time is set, the difference estimator outputs an estimation for \((F(S_t) - F(S_{e}))\) for all times \( t > e \) (provided some conditions on that difference). That is, once the difference estimator’s enabling time is set, it cannot be changed. And so, if an estimation is needed for some other enabling time, say \( e' \neq e \), then an additional instance of a difference estimator is needed. Our framework requires from such an estimator to be able to provide estimations for multiple enabling times, as long as the estimation periods do not overlap. This is captured in the following definition.

Definition 2.3 (Toggle Difference Estimator). Let \( F : ([n] \times \mathbb{Z})^* \rightarrow \mathbb{R} \) be a function, and let \( m, p \in \mathbb{N} \) and \( \gamma, \alpha, \delta \in (0, 1) \) be parameters. Let \( E \) be an algorithm with the following syntax. In every time step \( t \in [m] \), algorithm \( E \) obtains an update \((s_t, \Delta_t, b_t) \in ([n] \times \mathbb{Z} \times \{0, 1\}) \) and outputs a number \( z_t \). Here \((s_t, \Delta_t)\) denotes the current update, and \( b_t \) is an indicator for when the current time \( t \) should be considered as the new enabling time. We consider input streams \( S \in ([n] \times \mathbb{Z} \times \{0, 1\})^m \) such that there are at most \( p \) time steps \( t \) for which \( b_t = 1 \), and denote these time steps as \( 1 \leq e^1 < e^2 < \cdots < e^p < m \). Also, for a time step \( t \in [m] \) we denote \( e(t) = \max\{e^i : e^i \leq t\} \).

Algorithm \( E \) is a \((\gamma, \alpha, p, \delta)\)-toggle difference estimator for \( F \) if the following holds for every such input stream \( S \). With probability at least \( 1 - \delta \), for every \( t \in [m] \) such that

\[ |F(S_t) - F(S_{e(t)})| \leq \gamma \cdot F(S_{e(t)}) \quad (1) \]

the algorithm outputs a value \( z_t \) such that \( z_t \in (F(S_t) - F(S_{e(t)})) \pm \alpha \cdot F(S_{e(t)}) \).

This definition generalizes the notion of a difference estimator (DE) from [29], in which \( p = 1 \). In Section 4 we show that this extension comes at a very low cost in terms of the space complexity. Note that on times \( t \) s.t. the requirements specified w.r.t. \( \gamma \) do not hold, there is no accuracy guarantee from the TDE algorithm.

---

5 For an integer \( n \in \mathbb{N} \) denote \([n] = \{0, 1, \ldots, n - 1\} \) (that is \(|[n]| = n\)).
2.1 Preliminaries from Differential Privacy

Differential privacy [13] is a mathematical definition for privacy that aims to enable statistical analyses of databases while providing strong guarantees that individual-level information does not leak. Consider an algorithm $A$ that operates on a database in which every row represents the data of one individual. Algorithm $A$ is said to be **differentially private** if its outcome distribution is insensitive to arbitrary changes in the data of any single individual. Intuitively, this means that algorithm $A$ leaks very little information about the data of any single individual, because its outcome would have been distributed roughly the same even if without the data of that individual. Formally,

▶ **Definition 2.4** ([13]). Let $A$ be a randomized algorithm that operates on databases. Algorithm $A$ is $(\varepsilon, \delta)$-differentially private if for any two databases $S, S'$ that differ on one row, and any event $T$, we have

$$\Pr[A(S) \in T] \leq e^\varepsilon \cdot \Pr[A(S') \in T] + \delta.$$ 

3 A Framework for Adversarial Streaming

Our transformation from an oblivious streaming algorithm $E_{ST}$ for a function $F$ into an adversarially robust algorithm requires the following two conditions.

1. The existence of a toggle difference estimator $E_{TDE}$ for $F$, see Definition 2.3.
2. Every single update can change the value of $F$ up to a factor of $(1 \pm \alpha')$ for some $\alpha' = O(\alpha)$. Formally, throughout the analysis we assume that for every stream $S$ and for every update $u = (s, \Delta)$ it holds that

$$(1 - \alpha')F(S) \leq F(S, u) \leq (1 + \alpha')F(S).$$

▶ **Remark 3.1.** These conditions are identical to the conditions required by [29]. Formally, they require only a difference estimator instead of a toggle difference estimator, but we show that these two objects are equivalent. See Section 4.

▶ **Remark 3.2.** Condition 2 can be met for many functions of interest, by applying our framework on portions of the stream during which the value of the function is large enough. For example, when estimating $L_2$ with update weights $\pm 1$, whenever the value of the function is at least $\Omega(1/\alpha)$, a single update can increase the value of the function by at most a $(1 + \alpha)$ factor. Estimating $L_2$ whenever the value of the function is smaller than $O(1/\alpha)$ can be done using an existing (oblivious) streaming algorithm with error $\rho = O(\alpha)$. To see that we can use an oblivious algorithm in this setting, note that the additive error of the oblivious streaming algorithm is at most $O(\frac{\alpha}{\rho}) \ll 1$. Hence, by rounding the answers of the oblivious algorithm we ensure that its answers are exactly accurate (rather than approximate). As the oblivious algorithm returns exact answers in this setting, it must also be adversarially robust.

3.1 Construction Overview

Our construction builds on the constructions of [29] and [19]. At a high level, the structure of our construction is similar to that of [29], but our robustness guarantees are achieved using differential privacy, similarly to [19], and using our new concept of TDE.

Our algorithm can be thought of as operating in **phases**. In the beginning of every phase, we aggregate the estimates given by our strong trackers with differential privacy, and “freeze” this aggregated estimate as the base value for the rest of the phase. Inside every
phase, we privately aggregate (and “freeze”) estimates given by our TDE’s. More specifically, throughout the execution we aggregate TDE’s of different types/levels (we refer to the level that is currently being aggregated as the active level). At any point in time we estimate the (current) value of the target function by summing specific “frozen” differences together with the base value.

We remark that, in addition to introducing the notion of TDE’s, we had to incorporate several modifications to the framework of [29] in order to make it compatible with our TDE’s and with differential privacy. In particular, [29] manages phases by placing fixed thresholds (powers of 2) on the value of the target function; starting a new phase whenever the value of the target function crosses the next power of 2. If, at some point in time, the value of the target function drops below the power of 2 that started this phase, then this phase ends, and they go back to the previous phase. This is possible in their framework because the DE’s of the previous phase still exist in memory and are ready to be used. In our framework, on the other hand, we need to share all of the TDE’s across the different phases, and we cannot go back to “TDE’s of the previous phase” because these TDE’s are now tracking other differences. We overcome this issue by modifying the way in which differences are combined inside each phase.

In Algorithm 1 we present a simplified version of our main construction, including inline comments to improve readability. The complete construction is given in Algorithm RobustDE.

3.2 Analysis Overview

At a high level, the analysis can be partitioned into five components (with one component being significantly more complex than the others). We now elaborate on each of these components; see the full version of this work for the formal analysis [4].

3.2.1 First component: Privacy analysis

We show that our construction satisfies differential privacy w.r.t. the collection of random strings on which the oblivious algorithms operate. Recall that throughout the execution we aggregate (with differential privacy) the outcome of our estimators from the different levels. Thus, in order to show that the whole construction satisfies privacy (using composition theorems) we need to bound the maximal number of times we aggregate the estimates from the different levels. However, we can only bound this number under the assumption that the framework is accurate (in the adaptive setting), and for that we need to rely on the privacy properties of the framework. So there is a bit of circularity here. To simplify the analysis, we add to the algorithm hardcoded caps on the maximal number of times we can aggregate estimates at the different levels. This makes the privacy analysis straightforward. However, we will later need to show that this hardcoded capping “never” happens, as otherwise the algorithm fails. These hardcoded caps are specified by the parameters $P_j$ (both in the simplified algorithm and in the complete construction), using which we make sure that the estimators at level $j$ are never aggregated (with differential privacy) more than $P_j$ times.

---

6 We remark that as the hardcoded capping “never” happens, we can in fact remove it from the algorithm. One way or another, however, we must derive a high probability bound on the number of times we can aggregate estimates at the different levels.
Algorithm 1: Simplified presentation of RobustDE.

**Input:** Stream \( S = \{ (s_t, \Delta t) \}_{t \in [m]} \), accuracy parameter \( \alpha \), and a bound on the flip number \( \lambda \).

**Estimators used:** Strong tracker \( E_{ST} \) and toggle-difference-estimator \( E_{TDE} \) for \( F \).

**Initialization.** Let \( \beta = \lceil \log(\alpha^{-1}) \rceil \) denote the number of levels and let \( P_j = O(2^{-j} \lambda) \). For every level \( j \in \{0, \ldots, \beta - 1\} \) initialize \( O(\sqrt{P_j}) = O(\sqrt{2^{-j} \lambda}) \) copies of the TDE algorithm \( E_{TDE} \) with parameter \( \gamma = O(2^j \alpha) \) and set their initial estimation to be \( Z_j = 0 \). Also initialize \( O(\sqrt{P_\beta}) = O(\sqrt{\alpha \lambda}) \) copies of the strong tracker \( E_{ST} \), and set their initial estimation to be \( Z_{ST} = 0 \). (We denote \( j = \beta \) for the level of the strong trackers, and denote \( Z_\beta = Z_{ST} \).)

Initialize counter \( \tau \).

% As the execution progresses, we update the variables \( Z_j \), in which we maintain aggregations of the estimates given by our strong trackers and difference estimators. We make sure that, at any point in time, we can compute an estimation for the target function by carefully combining the values of these variables. This careful combination is handled using the counter \( \tau \).

% It is convenient to partition the time points into phases, where during each phase the value of \( Z_{ST} \) remains constant. Intuitively, in the beginning of every phase we compute a very accurate estimation for the target function using \( E_{ST} \), and then in the rest of the phase, we augment that estimation with weaker estimations given by our TDE’s (and accumulate errors along the way). Our error is reset at the beginning of every phase.

For every time step \( t \in [m] \):

1. Get the update \( (s_t, \Delta t) \) and feed it to all estimators.
2. Select the relevant estimator level according to \( \tau \) into \( j \) (where \( j = \beta \) in case the relevant level is that of the strong trackers, which happens only if \( \tau = O(1/\alpha) \)).
3. Let \( Z \) denote the sum of previously computed aggregates for levels \( i > j \) such that the \( i \)th bit of \( \tau \) is 1 (in binary representation). That is, \( Z = \sum_{i > j : \tau[i] = 1} Z_i \).
   % Note that if \( j = \beta \) then \( Z = 0 \).
4. Summed with \( Z \), every estimator from level \( j \) suggests an estimation for \( F(t) \). If the previous output \( \text{Out} \) is “close enough” to (most of) these suggestions, or if level \( j \) has already been aggregated \( P_j \) times, then goto Step 5. Otherwise, modify \( \text{Out} \) as follows.

   a. \( Z_j \leftarrow \) Differentially private approximation for the median of the outputs given by the estimators at level \( j \).
   b. Re-enable all TDE’s at levels \( i < j \).
   % That is, from now on, all TDE’s at levels \( i < j \) are estimating differences from future time steps to the current time step.
   c. If \( j = \beta \), then set \( \tau \leftarrow 0 \). Otherwise set \( \tau \leftarrow \tau + 1 \).
   % That is, if \( j = \beta \), which happens only if \( \tau = O(1/\alpha) \), then we start a new phase.
   d. \( \text{Out} \leftarrow Z + Z_j \).
5. Output \( \text{Out} \)
3.2.2 Second component: Conditional accuracy

We show that if the following two conditions hold, then the framework is accurate:

**Condition (1):** At any time step throughout the execution, at least 80% of the estimators in every level are accurate (w.r.t. the differences that they are estimating).

**Condition (2):** The hardcoded capping never happens.

This is the main technical part in our analysis; here we provide an oversimplified overview, hiding many of the technicalities. We first show that if Conditions (1) and (2) hold then the framework is accurate. We show this by proving a sequence of lemmas that hold (w.h.p.) whenever Conditions (1) and (2) hold. We now elaborate on some of these lemmas. Recall that throughout the execution we “freeze” aggregated estimates given by the different levels. The following lemma shows that these “frozen” aggregations are accurate (at the moments at which we “freeze” them). This Lemma follows almost immediately from Condition (1), as if the vast majority of our estimators are accurate, then so is their private aggregation.

▶ **Lemma 3.3** (informal version of Lemma A.2). In every time step \( t \in [m] \) in which we compute a value \( Z_j \) (in Step 14a of Algorithm RobustDE, or Step 4a of the simplified algorithm) it holds that \( Z_j \) is accurate. Informally, if the current level \( j \) is that of the strong trackers, then \( |Z_j - F(t)| < \alpha \cdot F(t) \), and otherwise \( |Z_j - (F(t) - F(e_j))| < \alpha \cdot F(e_j) \), where \( e_j \) is the last enabling time of level \( j \).

During every time step \( t \in [m] \), we test whether the previous output is still accurate (and modify it if it is not). This test is done by comparing the previous output with (many) suggestions we get for the current value of the target function. These suggestions are obtained by summing the outputs of the estimators at the currently active level \( j \) together with a (partial) sum of the previously frozen estimates (denoted as \( Z \)). This is done in Step 14 of Algorithm RobustDE, or in Step 4 of the simplified algorithm. The following lemma, which we prove using Lemma 3.3, states that the majority of these suggestions are accurate (and hence our test is valid).

▶ **Lemma 3.4** (informal version of Lemma A.4). Fix a time step \( t \in [m] \), and let \( j \) denote the level of active estimators. Then, for at least 80% of the estimators in level \( j \), summing their output \( z \) with \( Z \) is an accurate estimation for the current value of the target function, i.e., \( |F(t) - (Z + z)| \leq \alpha \cdot F(t) \).

So, in every iteration we test whether our previous output is still accurate, and our test is valid. Furthermore, when the previous output is not accurate, we modify it to be \((Z + Z_j)\), where \( Z_j \) is the new aggregation (the new “freeze”) of the estimators at level \( j \). So this modified output is accurate (assuming that the hardcoded capping did not happen, i.e., Condition (2), as otherwise the output is not modified). We hence get the following lemma.

▶ **Lemma 3.5** (informal version of Lemma A.5). In every time step \( t \in [m] \) we have

\[
|\text{Output}(t) - F(t)| \leq \alpha \cdot F(t).
\]

That is, the above lemma shows that our output is “always” accurate. Recall, however, that this holds only assuming that Conditions (1) and (2) hold.

3.2.3 Third component: Calibrating to avoid capping

We derive a high probability bound on the maximal number of times we will aggregate estimates at the different levels. In other words, we show that, with the right setting of parameters, we can make sure that Condition (2) holds. The analysis of this component still assumes that Condition (1) holds.
We first show that between every two consecutive times in which we modify our output, the value of the target function must change noticeably. Formally,

\[ \text{Lemma 3.6 (informal version of Lemma A.6). Let } t_1 < t_2 \in [m] \text{ be consecutive times in which the output is modified (i.e., the output is modified in each of these two iterations, and is not modified between them). Then, } |F(t_2) - F(t_1)| = \Omega (\alpha \cdot F(t_1)). \]

We leverage this lemma in order to show that there cannot be too many time steps during which we modify our output. We then partition these time steps and “charge” different levels \( j \) for different times during which the output is modified. This allows us to prove a probabilistic bound on the maximal number of times we aggregate the estimates from the different levels (each level has a different bound). See Lemma A.7 for the formal details.

### 3.2.4 Forth component: The framework is robust

We prove that Condition (1) holds (w.h.p.). That is, we show that at any time step throughout the execution, at least 80\% of the estimators in every level are accurate.

This includes two parts. First, in Lemma A.8, we show that throughout the execution, the condition required by our TDE’s hold (specifically, see 1 in Definition 2.3). This means that, had the stream been fixed in advance, then (w.h.p.) we would have that all of the estimators are accurate throughout the execution. In other words, this shows that if there were no adversary then (a stronger variant of) Condition (1) holds.

Second, in Lemma A.9 we leverage the generalization properties of differential privacy to show that Condition (1) must also hold in the adversarial setting. This lemma is similar to the analysis of [19].

### 3.2.5 Fifth component: Calculating the space complexity

In the final part of the analysis, we calculate the total space needed by the framework by accounting for the number of estimators in each level (which is a function of the high probability bound we derived on the number of aggregations done in each level), and the space they require.

### 4 Toggle Difference Estimator from a Difference Estimator

We present a simple method that transforms any difference estimator to a toggle difference estimator. The method works as follows. Let DE be a difference estimator (given as an subroutine). We construct a TDE that instantiates two copies of the given difference estimator: \( \text{DE}_{\text{enable}} \) and \( \text{DE}_{\text{fresh}} \). It also passes its parameters, apart of the enabling times, verbatim to both copies. As DE is set to output estimations only after receiving an (online) enabling time \( e \), the TDE never enables the copy \( \text{DE}_{\text{fresh}} \). Instead, \( \text{DE}_{\text{fresh}} \) is used as a fresh copy that received the needed parameters and the stream \( S \) and therefore it is always ready to be enabled. Whenever a time \( t \) is equal to some enabling time (i.e. \( t = e^i \) for some \( i \in [p] \)), then the TDE copies the state of \( \text{DE}_{\text{fresh}} \) to \( \text{DE}_{\text{enable}} \) (running over the same space), and then it enables \( \text{DE}_{\text{enable}} \) for outputting estimations.

\[ \text{Corollary 1. For any function } F, \text{ provided that there exist a } (\gamma, \alpha, \delta)-\text{Difference Estimator for } F \text{ with space } S_{\text{DE}}(\gamma, \alpha, \delta, n, m), \text{ then there exists a } (\gamma, \alpha, \delta, p)-\text{Toggle Difference Estimator for } F \text{ with space } S_{\text{TDE}}(\gamma, \alpha, \delta, p, n, m) = 2 \cdot S_{\text{DE}}(\gamma, \alpha, \delta/p, n, m). \]
Note that for a DE whose space dependency w.r.t. the failure parameter \( \delta \) is logarithmic, the above construction gives a TDE with at most a logarithmic blowup in space, resulting from the \( p \) enabling times.

5 Applications

Our framework is applicable to functionalities that admit a strong tracker and a difference estimator. As [29] showed, difference estimators exist for many functionalities of interest in the insertion only model, including estimating frequency moments of a stream, estimating the number of distinct elements in a stream, identifying heavy-hitters in a stream and entropy estimation. However, as we mentioned, we are not aware of non-trivial DE constructions in the turnstile model. In more detail, [29] presented DE for the turnstile setting, but these DE require additional assumptions and do not exactly fit our framework (nor the framework of [29]).

To overcome this challenge we introduce a new monitoring technique which we use as a wrapper around our framework. This wrapper allows us to check whether the additional assumptions required by the DE hold, and reset our system when they do not. As a concrete application, we present the resulting bounds for \( F_2 \) estimation.

**Definition 5.1** (Frequency vector). Let \( S = ((s_1, \Delta_1), \ldots, (s_m, \Delta_m)) \in ([n] \times \{\pm 1\})^m \) be a stream. The frequency vector of the stream \( S \) is the vector \( u \in \mathbb{Z}^n \) whose \( i \)-th coordinate is \( u[i] = \sum_{j \in [m], s_j = i} \Delta_j \). We write \( u^{(t)} \) to denote the frequency vector of the stream \( S_t \), i.e., restricted to the first \( t \) updates. Given two time points \( t_1 \leq t_2 \in [m] \) we write \( u^{(t_1,t_2)} \) to denote the frequency vector of the stream \( S_{t_1}^{t_2} \), i.e., restricted to the updates between time \( t_1 \) and \( t_2 \).

In this section we focus on estimating \( F_2 \), the second moment of the frequency vector. That is, after every time step \( t \), after obtaining the next update \( (s_t, \Delta_t) \in ([n] \times \{\pm 1\}) \), we want to output an estimation for

\[
\|u^{(t)}\|_2^2 = \sum_{i=1}^n |u^{(t)}[i]|^2.
\]

Woodruff and Zhou [29] presented a \((\gamma, \alpha, \delta)\)-difference estimator for \( F_2 \) that works in the turnstile model, under the additional assumption that for any time point \( t \) and enabling time \( e \leq t \) it holds that

\[
\|u^{(e,t)}\|_2^2 \leq \gamma \cdot \|u^{(e)}\|_2^2.
\]

(2)

In general, we cannot guarantee that this condition holds in a turnstile stream. To bridge this gap, we introduce the notion of twist number (see Definition 1.6) in order to control the number of times during which this condition does not hold (when this condition does not hold we say that a violation has occurred). Armed with this notion, our approach is to run our framework (algorithm RobustDE) alongside a validation algorithm (algorithm Guardian) that identifies time steps at which algorithm RobustDE loses accuracy, meaning that a violation has occurred. We then restart algorithm RobustDE in order to maintain accuracy. As we show, our notion of twist number allows us to bound the total number of possible violation, and hence, bound the number of possible resets. This in turn allows us to bound the necessary space for our complete construction. Here we only state the result; see the full version of this work [4] for the details.
Theorem 5.2. There exists an adversarially robust $F_2$ estimation algorithm for turnstile streams of length $m$ with a bounded $(O(\alpha), m)$-flip number and $(O(\alpha), m)$-twist number with parameters $\lambda$ and $\mu$ correspondingly, that guarantees $\alpha$-accuracy with probability at least $1 - 1/m$ in all time $t \in [m]$ using space complexity of 

$$\tilde{O}\left(\frac{\sqrt{\alpha \lambda} + \mu}{\alpha^2} \log^{3.5}(m)\right).$$

As we mentioned, this should be contrasted with the result of [19], who obtain space complexity $O\left(\frac{\sqrt{\lambda}}{\alpha^2} \log^{3.5}(m)\right)$ for robust $F_2$ estimation in the turnstile setting. Hence, our new result is better whenever $\mu \ll \lambda$. 

References

1. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In SODA, pages 459–467, 2012. doi:10.1137/1.9781611973099.40.
2. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In PODS, pages 5–14, 2012. doi:10.1145/2213556.2213560.
3. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. Journal of Computer and system sciences, 58(1):137–147, 1999.
4. Idan Attias, Edith Cohen, Moshe Shechner, and Uri Stemmer. A framework for adversarial streaming via differential privacy and difference estimators. CoRR, abs/2107.14527, 2021. arXiv:2107.14527.
5. Raef Bassily, Kobbi Nissim, Adam D. Smith, Thomas Steinke, Uri Stemmer, and Jonathan R. Ullman. Algorithmic stability for adaptive data analysis. SIAM J. Comput., 50(3), 2021.
6. Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer. Dynamic algorithms against an adaptive adversary: generic constructions and lower bounds. In STOC, pages 1671–1684. ACM, 2022.
7. Omri Ben-Eliezer, Rajesh Jayaram, David P Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms via differential privacy and difference estimators. CoRR, abs/2107.14527, 2021. arXiv:2107.14527.
8. Vladimir Braverman, Avinatan Hassidim, Yossi Matias, Mariano Schain, Sandeep Silwal, and Samson Zhou. Adversarial robustness of streaming algorithms through importance sampling. CoRR, abs/2106.14952, 2021.
9. Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In STOC, pages 117–126. ACM, 2015.
10. Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Theory of cryptography conference, pages 265–284. Springer, 2006.
11. A. C. Gilbert, B. Hemenway, A. Rudra, M. J. Strauss, and M. Wootters. Recovering simple signals. In 2012 Information Theory and Applications Workshop, pages 382–391, 2012.
12. A. C. Gilbert, B. Hemenway, M. J. Strauss, D. P. Woodruff, and M. Wootters. Reusable low-error compressive sampling schemes through privacy. In 2012 IEEE Statistical Signal Processing Workshop (SSP), pages 536–539, 2012.
A Framework for Adversarial Streaming

Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. Adaptive machine unlearning. arXiv preprint, 2021. arXiv:2106.04378.

Moritz Hardt and Jonathan R. Ullman. Preventing false discovery in interactive data analysis is hard. In FOCS, pages 454–463. IEEE Computer Society, 2014.

Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In STOC, pages 121–130, 2013.

Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. In NeurIPS, 2020.

Christopher Jung, Katrina Ligett, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Moshe Shenfeld. A new analysis of differential privacy’s generalization guarantees. In ITCS, volume 151 of LIPIcs, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. Separating adaptive streaming from oblivious streaming using the bounded storage model. In CRYPTO 2021.

Aryeh Kontorovich, Menachem Sadigurschi, and Uri Stemmer. Adaptive data analysis with correlated observations. In ICML, volume 162 of Proceedings of Machine Learning Research, pages 11483–11498. PMLR, 2022.

Ilya Mironov, Moni Naor, and Gil Segev. Sketching in adversarial environments. SIAM J. Comput., 40(6):1845–1870, 2011. doi:10.1137/080733772.

Kobbi Nissim, Adam D. Smith, Thomas Steinke, Uri Stemmer, and Jonathan Ullman. The limits of post-selection generalization. In NeurIPS, pages 6402–6411, 2018.

Kobbi Nissim and Uri Stemmer. Concentration bounds for high sensitivity functions through differential privacy. J. Priv. Confidentiality, 9(1), 2019.

Moshe Shenfeld and Katrina Ligett. A necessary and sufficient stability notion for adaptive generalization. In NeurIPS, pages 11481–11490, 2019.

Moshe Shenfeld and Katrina Ligett. Generalization in the face of adaptivity: A bayesian perspective. CoRR, abs/2106.10761, 2021. arXiv:2106.10761.

Thomas Steinke and Jonathan R. Ullman. Interactive fingerprinting codes and the hardness of preventing false discovery. In COLT, volume 40 of JMLR Workshop and Conference Proceedings, pages 1588–1628. JMLR.org, 2015.

David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. In FOCS, 2021.

A Missing Statements

In this section we provide the formal versions of the informal statements that appear in the main body of this paper. Due to space restrictions, the analysis of these statements is deferred to the full version of this work [4].

Assumption A.1 (Accurate estimations). Fix a time step $t \in [m]$. Let $j \in [\beta + 1] \cup \{W\}$ be a level of estimators. Recall that $K_j$ denotes the number of the estimators in level $j$, and let $z^j_1, \ldots, z^j_{K_j}$ denote the estimations given by these estimators. Then:

1. For $j \in \{\beta, W\}$, $|\{k \in [K_j] : |z^j_k - F(t)| < \alpha_{ST} \cdot F(t)\}| \geq (8/10)K_j$
2. For $j < \beta$, $|\{k \in [K_j] : |z^j_k - (F(t) - F(e_j))| < \alpha_{TDE} \cdot F(e_j)\}| \geq (8/10)K_j$

Lemma A.2 (Accuracy of frozen values). Let $t \in [m]$ be a time step such that

1. Assumption A.1 holds for every $t' \leq t$.
2. NoCapping = True during time $t$.
3. Algorithm PrivateMed was activated during time $t$ (on Step 14a).
Algorithm 2 RobustDE($\mathcal{S}, \alpha, \delta, \lambda, \mathcal{E}_{ST}, \mathcal{E}_{TDE}$).

Input: A stream $\mathcal{S} = \{\langle s_t, \Delta_t \rangle \}_{t \in [m]}$ accuracy parameter $\alpha$, failure probability $\delta$ and a bound on the flip number $\lambda$.

Estimators used: Strong tracker $\mathcal{E}_{ST}$ and toggle-difference-estimator $\mathcal{E}_{TDE}$ for the function $\mathcal{F}$.

Subroutines used: StitchFrozenVals($\tau$), ActiveLVL($\tau$).

Constants calculation:
1. StepSize($\alpha$) ← $O(\alpha)$, $\alpha_{ST}$ ← $O(\alpha)$, $\alpha_{TDE}$ ← $O(\alpha / \log(\alpha^{-1}))$, $\Gamma$ ← $\Theta(1)$.
2. Phase params: let PhaseSize ← $\lfloor \frac{1}{\text{StepSize}(\alpha)} \rfloor$, $\beta$ ← $\lfloor \log(\text{PhaseSize}) \rfloor$, $J$ ← $[\beta + 1] \cup \{W\}$.
3. For $j \in [\beta + 1]; P_j$ ← $O(2^{-j} \lambda)$, $P_W$ ← $P_{\beta}$; For $j \in [\beta]; \gamma_j$ ← $\Omega(2^j \alpha)$; For $j \in J; z_j = \tilde{O}(P_j^{0.5})$.
4. Estimator sets: For $j \in J$ set $K_j$ ← $\tilde{O}(z_j^{-1})$ and let $\hat{E}_j = \{E_j^k\}_{k \in [K_j]}$.

Initialization:
5. Start all estimators $\{\hat{E}_j\}_{j \in J}$.
6. Set thresholds: For $j \in J$, $T_j$ ← $K_j/2 + \text{Lap}(2 \cdot z_j^{-1})$.
7. For $j \in J$ set $Capp_j$ ← 0; For $j \in [\beta + 1]$ set $Z_j$ ← 0; $\tau$ ← $\text{PhaseSize}$.

For $t \in [m]$:
8. Get the update $\langle s_t, \Delta_t \rangle$ from $\mathcal{S}$ and feed $\langle s_t, \Delta_t \rangle$ into all estimators.
9. If $\{\{k \in [K_j] : z_j^k \notin \left(\frac{1}{1 - \Gamma} \cdot Z_{ST}, \Gamma \cdot Z_{ST}\right)\} + \text{Lap}(4 \cdot z_j^{-1}) | > T_W$ then set $\tau = 0$, redraw $T_W$.
10. $j$ ← $\text{ActiveLVL}(\tau)$
11. For $k \in [K_j]$ get estimation $z_j^k$ ← $E_j^k$ \hspace{1cm} % Estimation offset of ST
12. If $j = \beta$; $Z$ ← 0 \hspace{1cm} % Estimation offset of TDE
13. Else: $Z$ ← $\text{StitchFrozenVals}(\tau - 2^j + 1)$
14. If $\tau$ = 0 or $\left|\{k \in [K_j] : Z + z_j^k \notin \text{Output}(t - 1) \pm Z_{ST} \cdot \text{StepSize}(\alpha)\}\right| + \text{Lap}(4 \cdot z_j^{-1}) > T_j$
   a. $Z_j$ ← $\text{PrivateMed}\{z_j^k\}_{k \in [K_j]}$
   b. Redraw $T_j$
   c. For $j' \in [j], k \in [K_{j'}]$ set $e_j^{k_{j'}}$ ← $t$
   d. $\text{Capp}_j$ ← $\text{Capp}_j + 1$
   e. If $\tau[\beta] = 0$: $\tau$ ← $2^\beta$ \hspace{1cm} % Starting phase
   f. Elseif $\tau < 2^\beta + \text{PhaseSize}$: $\tau$ ← $\tau + 1$ \hspace{1cm} % Inner phase update
   g. If $\tau = 2^\beta + \text{PhaseSize}$: $\tau[\beta]$ ← $0$ \hspace{1cm} % Ending phase
15. NoCapping ← $\cap_{j \in J} \text{Capp}_j < P_j$
16. If NoCapping = True: $\text{Output}(t)$ ← $\text{StitchFrozenVals}(\tau)$
Let $j \in [\beta + 1]$ be the level of estimators used in time $t$. Let $Z_j$ be the value returned by \texttt{PrivateMed}, and suppose that $K_j = \Omega\left(\frac{1}{\varepsilon} \sqrt{P_j \cdot \log \left( \frac{1}{\varepsilon} \right)} \log \left( \frac{P_j}{\varepsilon^2 \alpha} \log(n) \right) \right)$. Then, with probability at least $1 - \delta^M/P_j$ we have that:

1. For $j = \beta$, $|Z_j - F(t)| < \alpha_{ST} \cdot F(t)$.
2. For $j < \beta$, $|Z_j - (F(t) - F(e_j))| < \alpha_{TDE} \cdot F(e_j)$.

\textbf{Definition A.3} (Good execution). Throughout the execution of algorithm \texttt{RobustDE}, for $j \in [\beta + 1] \cup \{W\}$ the algorithm draws at most $4m$ noises from Laplace distribution with parameter $\varepsilon_j$. In addition, denoting by $P_j$ for $j \in [\beta + 1]$ the number of times that algorithm \texttt{RobustDE} activates \texttt{PrivateMed} on estimations of level $j$. Denote $\delta_N = \delta/(4 \cdot (\beta + 2))$, $\delta^M = \delta/(4 \cdot (\beta + 1))$. Set the algorithm parameters as follows $K_j = \Omega\left(\frac{1}{\varepsilon} \log \left( \frac{P_j}{\varepsilon^2 \alpha} \log(n) \right) \right)$, $\varepsilon_j = O\left(\frac{\varepsilon}{\sqrt{P_j \cdot \log \left( \frac{1}{\varepsilon} \right)}} \right)$. We define a good execution as follows:

1. All noises for all types $j \in [\beta + 1] \cup \{W\}$ are at most $O\left(\frac{1}{\varepsilon} \log \left( \frac{m^2}{\varepsilon} \right) \right)$ in absolute value.
2. For all $j \in [\beta + 1]$, all first $P_j$ frozen values of level $j$ are accurate. That is, if $t$ is the time of the frozen value computation then:
   - For $j = \beta$, $|Z_j - F(t)| < \alpha_{ST} \cdot F(t)$.
   - For $j < \beta$, $|Z_j - (F(t) - F(e_j))| < \alpha_{TDE} \cdot F(e_j)$.

\textbf{Lemma A.4} (Estimation error). Let $t \in [m]$ be a time step such that

1. Assumption A.1 holds for every $t' \leq t$.
2. \texttt{NoCapping} = True during time $t$.

Let $j \in [\beta + 1]$ be the level of estimators used in time step $t$, and let $Z$ be the value computed in Step 13. Let $z_j^1, \ldots, z_j^K$ denote the estimations given by the estimators in level $j$. Then assuming a good execution (see Definition A.3), for at least $80\%$ of the indices $k \in [K_j]$ we have

$$|F(t) - (Z + z_j^k)| \leq \alpha_{Stitch} \cdot F(t),$$

where $\alpha_{Stitch} = \Gamma \cdot (\alpha_{ST} + \beta \cdot \alpha_{TDE})$.

\textbf{Lemma A.5} (Output accuracy). Let $t \in [m]$ be a time step such that

1. Assumption A.1 holds for every $t' \leq t$.
2. \texttt{NoCapping} = True during time $t$.

Then assuming a good execution (see Definition A.3) we have

$$|\text{Output}(t) - F(t)| \leq \alpha \cdot F(t),$$

provided that $\alpha_{ST} = O(\alpha)$, $\alpha_{TDE} = O(\alpha/\log(\alpha^{-1}))$ and for all $j \in [\beta + 1] \cup \{W\}$, $K_j = \Omega\left(\frac{1}{\varepsilon} \log \left( \frac{P_j}{\varepsilon^2 \alpha} \log(n) \right) \right)$. 

\begin{algorithm}
\caption{StitchFrozenVals($\tau$)}
\textbf{Input:} A counter $\tau$. Global Variables: $\alpha$, $Z_{ST}$, $Z_j$ for $j \in [\beta]$.
1. $FV \leftarrow \{j \in [\beta] | \tau[j] = 1\}$
2. Return $Z_{ST} + \sum_{j \in FV} Z_{TDE,j}$
\end{algorithm}

\begin{algorithm}
\caption{ActiveLVL($\tau$)}
\textbf{Input:} A counter $\tau$. Global parameter: $\beta$.
1. If $\tau[\beta] = 0$: Return $\beta$ % Selecting the ST level
2. Else Return The LSB of $(\tau + 1)$ % Selecting a TDE level
\end{algorithm}
Lemma A.6 (Function value progress between output-modifications). Let \( t_1 < t_2 \in [m] \) be consecutive times in which the output is modified (i.e., the output is modified in each of these two iterations, and is not modified between them), where
1. Assumption A.1 holds for every \( t' \leq t_2 \).
2. \text{NoCapping} = \text{True} during time \( t_2 \).
3. \( \tau \neq 0 \) during time \( t_2 \).

Then, assuming a good execution (see Definition A.3) we have:
\[
|F(t_2) - F(t_1)| \geq \text{StepSize}(\alpha) \cdot Z_{\text{ST}} - 2 \cdot \text{Stitch} \cdot \max\{F(t_1), F(t_2)\}
\]
\[
|F(t_2) - F(t_1)| \leq \text{StepSize}(\alpha) \cdot Z_{\text{ST}} + (2 \cdot \text{Stitch} + \text{MuSize}(\alpha)) \cdot \max\{F(t_1), F(t_2)\}
\]

Lemma A.7 (Output modifications of each level). Let \( S \) be the input stream of length \( m \) for algorithm \text{RobustDE} with a flip number \( \lambda_{\alpha'}(S) \) and let \( t \in [m] \) be a time step such that Assumption A.1 holds for every \( t' \leq t \).
Then, assuming a good execution (see Definition A.3), for every level \( j \in [\beta + 1] \cup \{W\} \) we have
\[
C_j(t) \leq O \left( \frac{\lambda_{\alpha'}(S)}{2^j} \right),
\]
where \( \alpha' = (1/2) \cdot \text{StepSize}(\alpha) = O(\alpha) \).

Lemma A.8 (bounded estimation ranges). Let \( t \in [m] \) be a time step such that
1. Level \( j \in [\beta] \) was selected (a TDE).
2. Assumption A.1 holds for every \( t' < t \).

Denote by \( e_j \) the last time step during which level \( j \) estimators were enabled. Then, assuming a good execution (see Definition A.3), the following holds:
\[
|F(t) - F(e_j)| \leq \gamma_j \cdot F(e_j)
\]
where \( \gamma_j = \frac{\alpha_j}{1 - \alpha_j} 2^{j+1} \cdot 2^j = O(2^j \cdot \alpha) \).

Lemma A.9 (Accurate Estimations (Lemma 3.2 [19])). The following holds for a good execution (see Definition A.3). Let \( t \in [m] \) be a time step such that:
1. Level \( j \) was selected.
2. Assumption A.1 holds for every \( t' < t \).

Let \( E(S, \pi) \) be the estimator of level \( j \) that was selected on time step \( t \), and let \( \pi \) be its (possibly dynamic) parameters. Let \( E(S, \pi) \) have (an oblivious) guarantee that all of its estimates are accurate with accuracy parameter \( \alpha_{\pi} \) with probability at least \( \frac{9}{10} \). Then, for sufficiently small \( \epsilon \), if algorithm \text{RobustDE} is \((\epsilon, \delta')\)-DP w.r.t. the random bits of the estimators \( \{E^k\}_{k \in K} \), then with probability at least \( 1 - \frac{1}{2^t} \), for time \( t \) we have:
1. For \( j \in [\beta, W] \), \[|\{k \in [K] : |z^k - F(t)| < \alpha_{\pi} \cdot F(t)\}| \geq (8/10)K\]
2. For \( j < \beta \), \[|\{k \in [K] : |z^k - (F(t) - F(e))| < \alpha_{\pi} \cdot F(e)\}| \geq (8/10)K\]

Where \( z^k \leftarrow E^k(S, \pi) \) for a set of size \( K \geq \frac{1}{2^t} \log \left( \frac{2^n}{\delta'} \right) \) of the oblivious estimator \( E(S, \pi) \).

Theorem A.10 (Framework for Adversarial Streaming - Space). Provided that there exist:
1. An oblivious streaming algorithm \( E_{\text{ST}} \) for functionality \( F \), that guarantees that with probability at least \( 9/10 \) all of it’s estimates are accurate to within a multiplicative error of \( 1 \pm \alpha_{\text{ST}} \) with space complexity of \( S_{\text{ST}}(\alpha_{\text{ST}}, n, m) \)
2. For every \( \gamma, p \) there is a \((\gamma, \alpha_{\text{TDE}}, \frac{1}{10})\)-TDE for \( F \) using space \( \gamma \cdot S_{\text{TDE}}(\alpha_{\text{TDE}}, n, m) \).

Then there exist an adversarially robust streaming algorithm for functionality \( F \) that for any stream with a bounded flip number \( \lambda_{\alpha'/s,m} < \lambda \), s.t. with probability at least \( 1 - \delta \) its output is accurate to within a multiplicative error of \( 1 \pm \alpha \) for all times \( t \in [m] \), and has a space complexity of
\[
O \left( \sqrt{\alpha \lambda} \cdot \text{polylog}(\alpha, \alpha^{-1}, \delta^{-1}, n, m) \right) 
\cdot \left[ S_{\text{ST}}(\alpha, n, m) + S_{\text{TDE}}(O(\alpha / \log(\alpha^{-1})), \alpha, n, m) \right].
\]