Truly Perfect Samplers for Data Streams and Sliding Windows

Rajesh Jayaram* David P. Woodruff† Samson Zhou‡

August 30, 2021

Abstract

In the G-sampling problem, the goal is to output an index i of a vector $f \in \mathbb{R}^n$, such that for all coordinates $j \in [n]$,

$$\mathbf{Pr}[i = j] = (1 \pm \epsilon)\frac{G(f_j)}{\sum_{k\in[n]} G(f_k)} + \gamma,$$

where $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is some non-negative function. If $\epsilon = 0$ and $\gamma = 1/\operatorname{poly}(n)$, the sampler is called *perfect*. In the data stream model, f is defined implicitly by a sequence of updates to its coordinates, and the goal is to design such a sampler in small space. Jayaram and Woodruff (FOCS 2018) gave the first perfect L_p samplers in turnstile streams, where $G(x) = |x|^p$, using $\operatorname{polylog}(n)$ space for $p \in (0, 2]$. However, to date all known sampling algorithms are not *truly perfect*, since their output distribution is only point-wise $\gamma = 1/\operatorname{poly}(n)$ close to the true distribution. This small error can be significant when samplers are run many times on successive portions of a stream, and leak potentially sensitive information about the data stream.

In this work, we initiate the study of *truly perfect* samplers, with $\epsilon = \gamma = 0$, and comprehensively investigate their complexity in the data stream and sliding window models. We begin by showing that sublinear space truly perfect sampling is impossible in the turnstile model, by proving a lower bound of $\Omega\left(\min\left\{n, \log\frac{1}{\gamma}\right\}\right)$ for any G-sampler with point-wise error γ from the true distribution. We then give a general time-efficient sublinear-space framework for developing truly perfect samplers in the insertion-only streaming and sliding window models. As specific applications, our framework addresses L_p sampling for all $p > 0$, e.g., $\tilde{\mathcal{O}}\left(n^{1-1/p}\right)$ space for $p \geq 1$, concave functions, and a large number of measure functions, including the $L_1 - L_2$, Fair, Huber, and Tukey estimators. The update time of our truly perfect L_p-samplers is $\mathcal{O}(1)$, which is an exponential improvement over the running time of previous perfect L_p-samplers.

1 Introduction

The streaming model of computation has emerged as an increasingly popular paradigm for analyzing massive data sets, such as network traffic monitoring logs, IoT sensor logs, financial market updates, e-commerce transaction logs, and scientific observations, as in computational biology, astronomy, or particle physics. In the (one-pass) streaming model, an underlying data set is implicitly defined

*Carnegie Mellon University and Google Research NYC. E-mail: rkjayara@google.com
†Carnegie Mellon University. E-mail: dwoodruf@cs.cmu.edu
‡Carnegie Mellon University. E-mail: samsonzhou@gmail.com

through sequential updates that arrive one-by-one and can only be observed once, and the proposed algorithms are required to use space that is sublinear in the size of the input.

Sampling has proven to be a fundamental and flexible technique for the analysis of massive data. A significant line of active work has studied sampling techniques [Vit85, GLH08, CCD11, CDK+11, CCD12, CDK+14, Haa16, Coh18] in big data applications such as network traffic analysis [GKMS01, EV03, MCS+06, HNG+07, TLJ10], database analysis [LNS90, HS92, LN95, HNSS96, GM98, Haa16, CG19], distributed computing [CMYZ10, TW11, CMYZ12, WZ16, JSTW19], and data summarization [FKV04, DV06, DRVW06, DV07, ADK09, MRWZ20]. Given a non-negative weight function $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$ and a vector $f \in \mathbb{R}^n$, the goal of a G-sampler is to return an index $i \in \{1, 2, \ldots, n\}$ with probability proportional to $G(f_i)$. In the data stream setting, the vector f, called the *frequency vector*, is given by a sequence of m updates (referred to as insertions or deletions) to its coordinates. More formally, in a data stream the vector f is initialized to 0^n, and then receives a stream of updates of the form $(i, \Delta) \in [n] \times \{-M, \ldots, M\}$ for some integer $M > 0$. The update (i, Δ) causes the change $f_i \leftarrow f_i + \Delta$. This is known as the *turnstile* model of streaming, as opposed to the *insertion-only* model where $\Delta \geq 0$ for all updates. A 1-pass G-sampler must return an index given only one pass through the updates of the stream.

The most well-studied weight functions are when $G(x) = |x|^p$ for some $p > 0$. Such samplers in their generality, known as L_p *samplers*, were introduced by [MW10]. L_p samplers have been used to develop efficient streaming algorithms for heavy hitters, L_p estimation, cascaded norm approximation, and finding duplicates [MW10, AKO11, JST11, BOZ12, JW18b, CPW20]. Formally, a G-sampler is defined as follows.

Definition 1.1 (G sampler)**.** *Let $f \in \mathbb{R}^n$, $0 \leq \epsilon, \gamma < 1$, $0 < \delta < 1$ and $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$ be a non-negative function satisfying $G(0) = 0$. An* $(\epsilon, \gamma, \delta)$-approximate G-sampler *is an algorithm that outputs an index $i \in [n]$ such that if $f \neq \vec{0}$, for each $j \in [n]$,*

$$\mathbf{Pr}\left[i = j\right] = (1 \pm \epsilon)\frac{G(f_j)}{\sum_{k \in [n]} G(f_k)} \pm \gamma \tag{1}$$

*and if $f = \vec{0}$, then the algorithm outputs a symbol $\perp$ with probability at least $1 - \delta$. The sampler is allowed to output **FAIL** with some probability δ, in which case it returns nothing.*

If $\epsilon = 0$ and $\gamma = n^{-c}$, where $c > 1$ is an arbitrarily large constant, then the sampler is called perfect*. If $\epsilon = 0$ and $\gamma = 0$, then the sampler is called truly perfect.*

In general, if $\epsilon > 0$ and $\gamma = n^{-c}$, where $c > 1$ is an arbitrarily large constant, an $(\epsilon, \gamma, \delta)$-sampler is commonly referred to as an ϵ-relative error *approximate sampler*. Notice that the guarantees on the distribution of a sampler are all conditioned on the sampler not outputting **FAIL**. In other words, conditioned on not outputting **FAIL**, the sampler must output a value in $[n] \cup \{\perp\}$ from a distribution which satisfies the stated requirements. When $f = \vec{0}$, the distribution in equation 1 is undefined; therefore the special symbol $\perp$ is needed to indicate this possibility.

In the case of L_p samplers with $p > 0$, the underlying distribution is given by $|f_j|^p / \|f\|_p^p$. Such samplers are particularly useful as subroutines for other streaming and data-analytic tasks. In the insertion-only model, the classical reservoir sampling technique of [Vit85] gives an $\mathcal{O}(\log n)$ space truly perfect L_1 sampling algorithm. However, when $p \neq 1$, or when negative updates are also allowed (i.e., the turnstile model), the problem becomes substantially more challenging. In fact, the question of whether such L_p samplers even exist using sublinear space was posed by Cormode, Murthukrishnan, and Rozenbaum [CMR05].

Monemizadeh and Woodruff partially answered this question by showing the existence of an $(\epsilon, n^{-c}, 1/2)$-approximate L_p sampler for $p \in [1,2]$ using poly $\left(\frac{c}{\epsilon}, \log n\right)$ bits of space in the turnstile model [MW10]. The space bounds were improved by Andoni, Krauthgamer, and Onak [AKO11], and then later by Jowhari, Saglam, and Tardos [JST11], to roughly $\mathcal{O}\left(c\epsilon^{-\max(1,p)} \log^2 n\right)$ for $p \in (0,2)$ and $\mathcal{O}\left(c\epsilon^{-2} \log^3 n\right)$ for $p = 2$. This matched the lower bound of $\Omega(\log^2 n)$ in terms of $\log n$ factors for $p < 2$, as shown in the same paper [JST11], but was loose in terms of ϵ. This gap was explained by Jayaram and Woodruff [JW18b], who gave the first *perfect* $(0, n^{-c}, 1/2)$-F_p samplers in the streaming model, using $\mathcal{O}\left(c \log^2 n\right)$ bits of space for $p \in (0,2)$ and $\mathcal{O}\left(c \log^3 n\right)$ bits of space for $p = 2$. For a further discussion on the development of L_p samplers in the streaming model, we direct the reader to the survey [CJ19]. In addition to L_p sampling, [CG19] also considered samplers for certain classes of concave functions G in the insertion-only model of streaming.

Truly Perfect Sampling. Unfortunately, none of the aforementioned perfect samplers are *truly perfect*. Specifically, they have an additive error of $\gamma = n^{-c}$, and space depending linearly on c. While this may be acceptable for some purposes where only a small number of samples are required, this error can have significant downstream consequences when many samplers are run independently. For instance, a common usage of sampling for network monitoring and event detection is to generate samples on successive portions of the stream, which are reset periodically (e.g., minute by minute). Additionally, in a large, distributed database, many independent samplers can be run locally on disjoint portions of the dataset. These samples can be used as compact summaries of the database, providing informative statistics on the distribution of data across machines. While the samples generated on a single portion of the data may be accurate enough for that portion, the $1/\operatorname{poly}(n)$ variation distance between the samples and true distribution accumulates over many portions. For large databases containing s distributed machines with $s \gg \operatorname{poly}(n)$, or for samplers run on short portions of high throughput streams, the resulting gap in variation distance between the joint distributions of the samples and the true distribution can blow up to a *constant*. This results in large accuracy issues for sensitive tests run on the data.

Moreover, this creates large issues for *privacy*, even when the identities of samples are anonymized. For instance, a non-truly perfect sampler may positively bias a certain subset $S \subset [n]$ of coordinates when a given entry is in the dataset (i.e, $x_i \neq 0$), and may negatively bias S if that entry is not present (i.e., $x_i = 0$). Given sufficiently many samples, an onlooker would be able to easily distinguish between these cases. Our truly perfect samplers achieve perfect security [Dat16], whereas in previous work a streaming algorithm that has input X could have an output that depends on arbitrary other information in X and thus could in principle reveal information about every other entry in X. If the entries of X correspond to sensitive data records, revealing as little as possible about X is crucial, and is the basis for studying perfect security.

Another important application of truly perfect sampling is in situations where samples from previous portions of the stream influence future portions of the stream, causing cascading blow-ups in error. For instance, samples and sketches can be used to approximate gradient updates for gradient descent [JZ13, ZZ15, NSW16, IRU+19], where a large number of future gradients naturally depend on the samples generated from prior ones. Unbiasedness is also important for interior point methods, since bias in estimates of the gradients can result in large drift, and therefore error, in the algorithm (see, e.g., Theorem 2 of [HPGS16]). Beyond non-adversarial adaptivity, we may also have a malicious attacker who uses adaptivity in an uncontrolled manner. For example, a malicious adversary can adaptively query a database for samples, with future queries depending on

past samples. Such streams with adaptive updates are the focus of the field of *adversarial robust streaming* [MBN+17, AMYZ19, BEY20, BEJWY20, HKM+20, WZ20, BHM+21, ACSS21]. Due to this adaptivity, the variation distance between the joint distributions can increase *exponentially*, causing large accuracy issues after only a small number of adaptive portions of the stream. Thus, even a perfect sampler would not avoid significant information leakage in such settings, and instead only a truly perfect sampler would be robust to drifts in the output distribution. Finally, truly perfect samplers are of fundamental importance in information-theoretic security.

The Sliding Window Model. While applicable for many situations in data analysis, the standard streaming model does not capture situations in which the data is considered time-sensitive. In applications such as network monitoring [CM05, CG08, Cor13], event detection in social media [OMM+14], and data summarization [CNZ16, ELVZ17], recent data is considered more accurate and important than data that arrived prior to a certain time window. To address such settings, [DGIM02] introduced the *sliding window model*, where only the W most recent updates to the stream induce the underlying input data, for some window size parameter $W > 0$. The most recent W updates form the *active data*, whereas updates previous to the W most recent updates are *expired*. The goal is to aggregate information about the active data using space sublinear in W. We remark that, generally speaking, the sliding window model is insertion-only by definition. Hence, the sliding window model is a strict generalization of the standard insertion-only streaming model.

The sliding window model is more appropriate than the unbounded streaming model in a number of applications [BBD+02, BOZ12, MM12, PGD15, WLL+16] and has been subsequently studied in a number of additional settings [LT06a, LT06b, BO07, DM07, BOZ12, BLLM15, BLLM16, BGL+18, BDM+20]. To date, however, no truly perfect, perfect, or even approximate L_p samplers for the sliding window model are known, leaving a substantive gap in our understanding of sampling for these models.

1.1 Our Contributions

In this work, we initiate the study of *truly perfect samplers*, for general weight functions G in the data stream and sliding window models. We begin by studying the problem of truly perfect sampling in the *turnstile* model of streaming, where both positive and negative updates can be made to the coordinates in f. We demonstrate that in the turnstile model, the additive $1/\operatorname{poly}(n)$ error in previous approximate and perfect L_p samplers is inherent [MW10, AKO11, JST11, JW18b].

Theorem 1.2. *Fix constant $\epsilon_0 < 1$, integer $r \geq 1$, and let $2^{-n/2} \leq \gamma < \frac{1}{4}$. Let $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$ be any function satisfying $G(x) > 0$ for $x \neq 0$, and $G(0) = 0$. Then any $(\epsilon_0, \gamma, \frac{1}{2})$-approximate G-sampler $\mathcal{A}$ in the r-pass turnstile streaming model must use $\Omega\left(\min\left\{n, \log\frac{1}{\gamma}\right\}\right)$ bits of space.*

Theorem 1.2 explains why all prior approximate and perfect samplers developed for the turnstile sliding window model paid a $1/\operatorname{poly}(n)$ additive error in their variation distance. In particular, when $\gamma = n^{-c}$, our lower bound of $\Omega(c \log n)$ for a $(\epsilon, \gamma, \frac{1}{2})$-$F_p$sampler for $p \in (0, 2]$ is nearly tight, given the upper bound of $\mathcal{O}\left(c \log^2 n\right)$ of [JW18b] for $p \in (0, 2)$ and $\mathcal{O}\left(c \log^3 n\right)$ for $p = 2$, which achieve perfect sampling ($\epsilon = 0$). This demonstrates that $\log\frac{1}{\gamma} \operatorname{polylog} n$ is the correct complexity of $(0, \gamma, \frac{1}{2})$-L_p sampling. Our lower bound is based on the fine-grained hardness of the EQUALITY problem from two-party communication complexity, demonstrating that a $(\epsilon, \gamma, 1/2)$-G sampler yields a communication protocol which solves EQUALITY with γ advantage.

Given the strong impossibility results for designing truly perfect samplers in the turnstile model, we shift our attention to the fundamental insertion-only model. Given a measure function $G : \mathbb{R} \to \mathbb{R}^{\geq 0}$ such that $G(x) = G(-x)$, $G(0) = 0$ and G is non-decreasing in $|x|$, we define $F_G = \sum_{i=1}^{n} G(f_i)$. We design a general framework for designing truly perfect G-samplers for a large number of useful functions G in insertion-only streams and sliding windows with insertion-only updates. The framework is developed in Section 3, wherein several instantiations of the framework are given for specific functions G. Our theorem in its most general form is as follows, although we remark that for several applications, such as for F_p estimation, significant additional work is needed to apply the theorem.

Framework 1.3. *Let G be a function such that $0 \leq G(x) - G(x-1) \leq \zeta$ for all $x \geq 1$. For insertion-only streams, there exists a perfect G sampler that succeeds with probability at least $1 - \delta$ and uses $\mathcal{O}\left(\frac{\zeta m}{F_G} \log n \log \frac{1}{\delta}\right)$ bits of space. For the sliding window model with insertion-only updates, there exists a truly perfect G sampler that succeeds with probability at least $1 - \delta$ and uses $\mathcal{O}\left(\frac{\zeta W}{F_G} \log^2 n \log \frac{1}{\delta}\right)$ bits of space. Further, the time to process each update is $\mathcal{O}(1)$ in expectation. (See Theorem 3.1.)*

The main barrier to applying Framework 1.3 to any arbitrary measure function G is obtaining a "good" lower bound $\widehat{F_G}$ to $F_G = \sum_{i \in [n]} G(f_i)$. Moreover, this lower bound must be obtained correctly with probability 1, as any possibility of failure of a randomized algorithm would necessarily contribute to additive error to the distribution of the samples, resulting in only a perfect, but not truly perfect, sampler.

Interestingly, our samplers utilize a timestamp-based reservoir sampling scheme, as opposed to the common *precision sampling* framework used for other L_p samplers [AKO11, JST11, JW18b, JW18a, JSTW19, CJLW20]. This property of our samplers makes our framework particularly versatile, and for instance allows it to be applied to the sliding window model of streaming.

As specific applications of our framework, we obtain the first truly perfect samplers for many fundamental sampling problems, including L_p sampling, concave functions, and a large number of measure functions, including the $L_1 - L_2$, Fair, Huber, and Tukey estimators. For $p \geq 1$, our results for L_p sampling are as follows; we defer $p \in (0, 1)$ to Theorem 3.3.

Theorem 1.4. *For the insertion-only streaming model and $p \geq 1$, there exists a truly perfect L_p sampler that uses $\mathcal{O}(1)$ update time and $\mathcal{O}\left(n^{1-1/p} \operatorname{polylog}(n)\right)$ bits of space.*

Together, Theorem 1.2 and Theorem 1.4 show a strong separation between turnstile and insertion-only truly perfect L_p samplers; surprisingly, for every $p > 1$, a truly perfect L_p sampler exists with $\mathcal{O}\left(n^{1-1/p} \operatorname{polylog}(n)\right)$ space in the insertion-only model, while in the turnstile model this requires $\Omega(n)$ space.

Another interesting feature of Theorem 1.4 is that the time to process each update is $\mathcal{O}(1)$! In contrast, the perfect samplers of [JW18b], which each are not truly perfect, have update time $n^{O(c)}$ to achieve variation distance $\frac{1}{n^c}$. Thus, we obtain an exponential improvement, and optimal running time.

Yet another interesting feature of our algorithm in Theorem 1.4 is that it is sampling-based rather than sketching-based, and thus if the indices have metadata associated with them then we can additionally return that metadata, whereas the sketching-based algorithm of [JW18b] cannot. For example, the indices may be keys we sample by and each key is part of some document; in this case we sample by the keys but additionally can return the document sampled.

Notice also that the complexity in Theorem 1.4 above degrades as $p \to 1$. For instance, for $p = 1$, our bound degrades to $n^{1-1/p} \log n = \log n$, which matches the complexity of reservoir sampling in the insertion only model, or the sliding window truly perfect L_1 sampler by [BOZ12].

M-Estimators. In addition to Theorem 1.4, Framework 1.3 also implies truly perfect samplers for a number of M-estimators:

- Truly perfect G samplers for insertion-only streams that use $\mathcal{O}\left(\log n \log \frac{1}{\delta}\right)$ bits of space when G is the $L_1 - L_2$ estimator, the Fair estimator, the Huber estimator, or the Tukey estimator. (See Corollary 3.6 and Theorem 5.4.)
- Truly perfect G samplers for sliding windows with insertion-only updates that use $\mathcal{O}\left(\log^2 n \log \frac{1}{\delta}\right)$ bits of space when G is the $L_1 - L_2$ estimator, the Fair estimator, or the Huber estimator, or the Tukey estimator. (See Corollary 4.2 and Theorem 5.5.)

Matrix Norms. Framework 1.3 can also be extended to truly perfect sampling for matrix norms. That is, given a matrix $\mathbf{M} \in \mathbb{R}^{n \times d}$, the goal is to sample a row $\mathbf{m}_i$ of $\mathbf{M}$ with probability proportional to $G(\mathbf{m}_i)$ for some given function G. For example, when $G(\mathbf{x}) = \sqrt{\sum_{i \in [d]} x_i^2}$ is the L_2 norm of each row, then such a row sampling primitive would be equivalent to $L_{1,2}$ sampling, which has recently been used in adaptive sampling techniques (see [MRWZ20] and references therein). See Theorem 3.7 for more details.

Turnstile Streams. Next, we show that Framework 1.3 can also be extended to strict turnstile streams, which combined with Theorem 1.2 shows the separation of general and strict turnstile streams. We give a general reduction as follows:

Theorem 1.5. *Suppose there exists a truly perfect L_p sampler in the one-pass insertion-only streaming model that uses S bits of space. Then there exists a truly perfect L_p sampler that uses $\tilde{\mathcal{O}}\left(Sn^{\gamma}\right)$ space and $\mathcal{O}\left(\frac{1}{\gamma}\right)$ passes over a strict turnstile stream, which induces intermediate frequency vectors with nonnegative coordinates.*

Truly Perfect F_0 Sampling. We give a truly perfect sampler for the important F_0 problem for both the insertion-only streaming model and the sliding window model (see Theorem 5.2 and Corollary 5.3). Our algorithm works by tracking the first $\sqrt{n}$ unique items in the stream to decide whether $F_0 > \sqrt{n}$. If $F_0 \leq \sqrt{n}$, then it suffices to output a random item among those collected. Otherwise, we simultaneously generate a set S of $\sqrt{n}$ random items, so that with constant probability, some item of S appears in the stream. We can then output any item of S that has appeared in the stream uniformly at random. Surprisingly, our algorithms use $\mathcal{O}\left(\sqrt{n} \log n\right)$ space whereas there exists a truly perfect F_0 sampler in the random oracle model with space $\mathcal{O}\left(\log n\right)$. We believe the complexity of truly perfect F_0 sampling without the assumption of a random oracle to be a fascinating open question.

Truly Perfect Sampling in the Random Order Model. Next, in Appendix C, we demonstrate that for *random order* streams, we can design a truly perfect L_2 sampling using only $\mathcal{O}\left(\log^2 n\right)$ bits of space. Since, as in our framework for truly perfect sampling, our algorithms

are timestamp based, they also apply to the more challenging sliding window model of streaming. The complexity of our sampler is a $\log n$ factor smaller than the complexity of the best known previous samplers [JW18b] in the adversarial order model, which had $\gamma = 1/\operatorname{poly}(n)$ additive error in their distribution, and which did not apply to the sliding window model. Our theorem for L_2 sampling on random order streams is as follows.

Theorem 1.6. *There exists a one-pass sliding window algorithm for random order insertion-only streams that outputs index $i \in [n]$ with probability $\frac{f_i^2}{F_2}$ and outputs **FAIL** with probability at most $\frac{1}{3}$, i.e., the algorithm is a truly perfect L_2 sampler, using $\mathcal{O}\left(\log^2 n\right)$ bits of space and $\mathcal{O}(1)$ update time.*

We generalize this approach to perfect L_p samplers on random order streams for integers $p > 2$.

Theorem 1.7. *Let $p > 2$ be a fixed integer. There exists a one-pass algorithm that outputs an index $i \in [n]$ with probability $\frac{f_i^p}{\sum_{j=1}^n f_j^p}$, and outputs **FAIL** with probability at most $\frac{1}{3}$ on a random-order insertion-only stream of length m , i.e., the algorithm is a truly perfect L_p sampler, using $\mathcal{O}\left(m^{1-\frac{1}{p-1}} \log n\right)$ bits of space and $\mathcal{O}(1)$ update time.*

For $p = 2$, intuitively our algorithm follows by tracking collisions between adjacent elements in the stream. Here, a collision occurs when to subsequent updates are made to the same coordinate $i \in [n]$. The probability that this occurs at a given timestep is $\frac{f_i(f_i-1)}{m(m-1)}$. Since this is not quite the right probability, we "correct" this distribution by a two part rejection sampling step to obtain a truly perfect sampler. For truly perfect L_p sampling on random order streams for integers $p > 2$, we store consecutive blocks of $m^{1-\frac{1}{p-1}}$ elements in the stream, along with their corresponding timestamps, and instead look for p-wise collisions within the block.

Fast Perfect L_p Sampling for $p < 1$. Lastly, we demonstrate that for insertion only streams, the runtime of the perfect L_p samplers of [JW18b] can be significantly improved. Specifically, these prior samplers had update time which was a large polynomial in n: roughly, to obtain a $(0, \gamma, 1/2)$ L_p-sampler, these algorithms required $\mathcal{O}\left(\frac{1}{\gamma}\right)$ runtime. This poses a serious barrier for usage of these perfect samplers in most applications. We demonstrate, however, that the tradeoff between runtime and distributional error in a perfect L_p sampler is not required for $p < 1$, by giving an algorithm with nearly tight space complexity, which achieves $\operatorname{poly}(\log n)$ update time. See Corollary B.11 for more details.

1.2 Our Techniques

Truly Perfect L_p Samplers. We begin by describing our sampling framework for the case of L_p samplers. Specifically, to obtain a truly perfect L_p sampler for insertion-only streams of length m and for $p \geq 1$, we run $\mathcal{O}\left(n^{1-1/p}\right)$ parallel instances of a single sampler, which uses only $\log n$ bits of space, but succeeds with probability $\Omega\left(\frac{1}{n^{1-1/p}}\right)$. Each instance of the single sampler first applies reservoir sampling to the updates in the stream to sample an item $s \in [n]$, along with a specific timestamp t_s when s was added to the reservoir sample, and keeps a counter c of how many times s appears in the stream after it is first sampled.

Inspired by a technique of Alon, Matias, and Szegedy for F_p-estimation [AMS99], we then prove that if c occurrences of s appear afterwards and we output s with probability proportional to $c^p - (c-1)^p$, then by a telescoping argument, the probability of outputting each $i \in [n]$ is proportional to $|f_i|^p$. Thus a sampler that successfully outputs a coordinate must do so from the desired distribution. To implement the rejection sampling step, we need to obtain a good normalizing factor so that the resulting step forms a valid distribution. We demonstrate that it suffices to estimate $\|f\|_\infty$ to obtain a good normalizing factor, which results in acceptance probability of at least $\Omega\left(\frac{1}{n^{1-1/p}}\right)$. We carry out this step deterministically with the Misra-Gries sketch, which is necessary since failure of any randomized estimation algorithm would introduce additive error into the sampler. By repeating $\mathcal{O}\left(n^{1-1/p}\right)$ times, we ensure that at least one sampler succeeds with constant probability, showing that we output **FAIL** with at most constant probability. We use a similar approach for $p \in (0, 1)$.

General Framework for Truly Perfect Sampler. The aforementioned telescoping argument for our truly perfect sampler for insertion-only updates can be viewed as using the identity $\sum_{c=1}^{f_i}(G(c)-G(c-1)) = G(f_i) - G(0)$ for $G(x) = x^p$. By the same reasoning, we can generalize the paradigm of "correcting" the sampling probability for any monotonic function G with $G(0) = 0$ by a subsequent rejection sampling step though the size of the acceptance probability depends on obtaining a good normalizing factor. For $F_G = \sum_{i=1}^{n} G(f_i)$, we show that we can similarly bound the probability each single sampler succeeds with probability roughly $\frac{F_G}{m}$. Thus we require roughly $\frac{m}{F_G}$ parallel instances of the single sampler.

For the case of the sliding window model, we can automatically expire the sample (s, t_s) as soon as t_s leaves the active window, causing an additional complexity in the sliding window model. However, by maintaining a number of "checkpoints", we can ensure that (s, t_s) is an active element with constant probability.

Random-Order Sampling. Finally, we improve our perfect L_p samplers for random-order insertion-only streams by using distributional properties of each stream update. Namely, we modify our timestamp based sampling scheme to randomly sample a number of p-tuples and search for collisions among the p-tuples. For $p = 2$, the idea is to consider two adjacent elements and see if they collide. Intuitively, an arbitrary position in the window is item i with probability $\frac{f_i}{m}$ due to the random order of the stream, where m is the length of the stream. The next update in the stream is also item i with probability $\frac{f_i-1}{m-1}$. Thus the probability that both the two positions are updates to coordinate i is $\frac{f_i(f_i-1)}{m(m-1)}$, which is not quite the right probability. Instead of using a telescoping argument as in the general framework, we instead "correct" this probability by sampling item i in a position with probability $\frac{1}{m}$. Otherwise, with probability $1 - \frac{1}{m}$, we sample item i if the item is in the next position as well. Now the probability of sampling i on the two adjacent elements is $\frac{1}{m}\frac{f_i}{m} + \frac{m-1}{m}\frac{f_i}{m}\frac{f_i-1}{m-1} = \frac{f_i^2}{m^2}$. We similarly generalize this argument to integer $p > 2$.

1.3 Preliminaries

We use $\mathbb{R}^{\geq 0}$ to denote a non-negative number. We use the notation $[n]$ to represent the set $\{1, \ldots, n\}$ for any positive integer n. We use $\text{poly}(n)$ to denote a fixed constant degree polynomial in n but we write $\frac{1}{\text{poly}(n)}$ to denote an arbitrary degree polynomial in n that can be determined from setting constants appropriately. When an event has probability $1 - \frac{1}{\text{poly}(n)}$ of occurring, we say the event

occurs with high probability. We similarly use polylog(n) to omit terms that are polynomial in $\log n$. We note that all space bounds in this paper are given in *bits*. In the insertion-only model of streaming, there is a vector $f \in \mathbb{R}^n$ which is initialized to the 0 vector. The stream then proceeds with a sequence of $m = \text{poly}(n)$ updates $i_1, i_2, \ldots, i_m$ (the exact value of m is unknown to the algorithm). After the first t time steps, the state of the vector, denoted $f^{(t)}$, is given by $f_j^{(t)} = \sum_{t' \leq t} \mathbf{1}(i_{t'} = j)$. In the sliding window model, at time step t only the most recent W updates form the active portion of the stream, so that in the sliding window model: $f_j^{(t)} \sum_{t' \in (t-W,t]} \mathbf{1}(i_{t'} = j)$.

2 Lower Bound for Truly Perfect Sampling in Turnstile Streams

In this section, we demonstrate that truly perfect G samplers cannot exist in sublinear space in the turnstile model. Specifically, we show that any perfect sampler with additive error $\gamma = n^{-c}$ requires space at least $\Omega(c \log n)$. This demonstrates that no sublinear space truly perfect sampler can exist in the turnstile model, and demonstrates the tightness (up to $\log n$ factors), of the previously known perfect and approximate L_p samplers [MW10, AKO11, JST11, JW18b].

Our lower bound is based on the fine-grained hardness of the EQUALITY problem from two-party communication complexity [BCK+14]. Specifically, consider the boolean function $\text{EQ}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ given by $\text{EQ}_n(x, y) = 1 \iff x = y$. In the two party, one way communication model, there are two parties: Alice and Bob. Alice is given a input string $x \in \{0,1\}^n$ and Bob is given $y \in \{0,1\}^n$. Then Alice must send a single message M to Bob, who must then output whether $\text{EQ}_n(x, y)$ correctly with some probability. A communication protocol $\mathcal{P}$ is a randomized two-party algorithm which takes the input (x, y) and decides on a message M and output procedure out for Bob given (M, y). The communication cost of $\mathcal{P}$ is denoted $\text{cost}(\mathcal{P}, x, y)$, and defined as the maximum number of bits sent in the message M over all coin flips of the algorithm, on inputs (x, y). We now define the randomized *refutation complexity* of a communication protocol for computing a Boolean function f. We define the *refutation cost*, *refutation error*, and *verification error* as:

$$\begin{aligned}
\text{rcost}(\mathcal{P}) &= \max_{(x,y) \in f^{-1}(0)} \text{cost}(\mathcal{P}, x, y) \\
\text{rerr}(\mathcal{P}) &= \max_{(x,y) \in f^{-1}(0)} \mathbf{Pr}\left[\text{out}(\mathcal{P}(x, y)) = 1\right] \\
\text{verr}(\mathcal{P}) &= \max_{(x,y) \in f^{-1}(1)} \mathbf{Pr}\left[\text{out}(\mathcal{P}(x, y)) = 0\right].
\end{aligned}$$

We define the randomized refutation complexity of a function f for an integer $r \geq 1$ as

$$R_{\epsilon,\delta}^{(r),\text{ref}}(f) = \min_{\mathcal{P}} \{\text{rcost}(\mathcal{P}) : \text{rerr}(\mathcal{P}) \leq \epsilon, \text{verr}(\mathcal{P}) \leq \delta\}$$

where the minimum is restricted to r-round communication protocols $\mathcal{P}$. Observe that the "trivial" one-round protocol for EQ_n achieves ϵ refutation error and communicates $\min(n, \log(1/\epsilon))$ bits, so it is as though the instance size drops from n to $\min(n, \log(1/\epsilon))$ when ϵ refutation error is allowed. Thus we define the effective instance size as

$$\hat{n} = \min\left\{n + \log(1-\delta),\ \log\left(\frac{(1-\delta)^2}{\epsilon}\right)\right\}.$$

Theorem 2.1 (Theorem 44 [BCK+14]). *We have* $R^{(r),\mathrm{ref}}_{\epsilon,\delta}(\textsc{EQ}_n) \geq \frac{1}{8}(1-\delta)^2(\hat{n}+\log(1-\delta)-5)$.

We show Theorem 1.2 by giving a reduction from EQUALITY and applying Theorem 2.1:

Theorem 1.2. *Fix constant* $\epsilon_0 < 1$, *integer* $r \geq 1$, *and let* $2^{-n/2} \leq \gamma < \frac{1}{4}$. *Let* $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$ *be any function satisfying* $G(x) > 0$ *for* $x \neq 0$, *and* $G(0) = 0$. *Then any* $(\epsilon_0, \gamma, \frac{1}{2})$*-approximate* G*-sampler* $\mathcal{A}$ *in the* r*-pass turnstile streaming model must use* $\Omega\left(\min\left\{n, \log\frac{1}{\gamma}\right\}\right)$ *bits of space.*

Proof. Given such a sampler $\mathcal{A}$, we give an algorithm for the two-party, r-round equality problem as follows. Alice is given $x \in \{0,1\}^n$, and creates a stream with frequency vector given by $f = x$. Bob then adds the vector $-y$ into the stream so that the final state of the frequency vector induced by the stream is $f = x - y$. Alice and Bob each run $\mathcal{A}$ on their stream and repeatedly pass the state of the algorithm between each other over the course of r rounds. Bob then finally obtains the output of the streaming algorithm $\mathcal{A}(f)$ after r rounds. If the output is **FAIL**, or anything except for the symbol $\perp$, then Bob declares $\textsc{EQ}_n(x,y) = 0$. If the output is $\perp$, Bob declares $\textsc{EQ}_n(x,y) = 1$. Notice by definition of a $(\epsilon_0, \gamma, \frac{1}{2})$-$G$ sampler (Definition 1.1), if we actually had $x = y$, then $f = \vec{0}$, so if $\mathcal{A}$ does not output **FAIL**, then it must declare $\perp$ with probability at least $1-\gamma$. Moreover, if $x \neq y$, then since $G((x-y)_i) > 0$ for some i, a correct sampler can output $\perp$ with probability at most γ. Furthermore, it can output **FAIL** in both cases with probability at most $\frac{1}{2}$.

The above protocol therefore satisfies that if $\textsc{EQ}_n(x,y) = 0$, Bob outputs 1 with probability at most γ, thus the refutation error is at most $\epsilon < \gamma$. Moreover, if $\textsc{EQ}_n(x,y) = 1$, then $\mathcal{A}$ outputs fail with probability $\frac{1}{2}$, and conditioned on not outputting fail it must output $\perp$ with probability at least $1-\gamma$. Thus, the verification error is at most $\delta < 1/2 + \gamma < 3/4$. Then we have $n - \log(1-\delta) > n/2$, and $\log(\frac{(1-\delta)^2}{\epsilon}) > \log(\frac{1}{16\gamma})$. Thus the effective input size is given by

$$\hat{n} > \min\left\{\frac{n}{2}, \log\frac{1}{16\gamma}\right\}$$

Thus, by Theorem 2.1, we have

$$\begin{aligned} R^{(r),\mathrm{ref}}_{\epsilon,\delta}(\textsc{EQ}_n) &\geq \frac{1}{8}(1-\delta)^2(\hat{n}+\log(1-\delta)-5) \\ &\geq \frac{1}{8\cdot 16}(\hat{n}-7) \\ &= \Omega(\hat{n}) \end{aligned} \tag{2}$$

which completes the proof of the lower bound. ∎

3 Framework for Truly Perfect Sampling

In this section, we first give a framework for truly perfect sampling for some measure function $G : \mathbb{R} \to \mathbb{R}^{\geq 0}$ such that $G(x) = G(-x)$, $G(0) = 0$ and G is non-decreasing in $|x|$. If we define $F_G = \sum_{i=1}^n G(f_i)$, then we say that a truly perfect G sampler outputs index $i \in [n]$ with probability $\frac{G(f_i)}{F_G}$. We then show how to apply the framework to L_p sampling where $G(x) = |x|^p$ and to various M-estimators, such as the $L_1 - L_2$, Fair, Huber, and Tukey estimators.

3.1 Algorithmic Framework

Our algorithm is based on running parallel instances of a single sampler. Each instance uses $\log n$ bits of space, but only succeeds with small probability and thus we need to run multiple instances to ensure that with sufficiently high probability, some instance succeeds. Each instance uses reservoir sampling to sample an item s and keeps a counter c of how many times s appears in the stream after it is sampled.

We first describe the Sampler algorithm. Given a stream of elements $u_1, \ldots, u_m$, where each $u_i \in [n]$, Sampler selects an index $j \in [m]$ uniformly at random and outputs u_j as well as the number of instances of u_j that appear after time j. The algorithm uses reservoir sampling to ensure that each item is selected with probability $\frac{1}{m}$. A counter is also maintained to track the number of instances of the sample. Each time a new sample replaces the existing sample in the reservoir sampling procedure, the counter is reset to zero.

Algorithm 1 Sampler: Reservoir sampling, counting number of times item has appeared afterwards.

Input: A stream of updates $u_1, u_2, \ldots, u_m$, where each $u_i \in [n]$ represents a single update to a coordinate of the underlying vector f.

Output: Sample each coordinate u_i with probability $\frac{1}{m}$ and output the number of occurrences that appears afterwards.

$s \leftarrow \emptyset$, $c \leftarrow 0$
for each update u_r **do**
  $s \leftarrow u_r$ with probability $\frac{1}{r}$
  if s is updated to u_r **then**
    $c \leftarrow 0$ ▷Reset counter.
  if $u_r = s$ **then**
    $c \leftarrow c + 1$ ▷Increment counter.
return s and c.

By outputting s with probability $\frac{G(c)-G(c-1)}{\zeta}$, where ζ is a parameter such that $G(x)-G(x-1) \leq \zeta$ for all possible coordinates x in the frequency vector, i.e., $x \in \{f_1, \ldots, f_n\}$, then it can be shown by a telescoping argument that the probability of outputting each $i \in [n]$ is "corrected" to roughly $\frac{G(f_i)}{\zeta \cdot m}$, where m is the length of the stream. Hence if the sampler successfully outputs a coordinate, it follows the desired distribution.

Algorithm 2 Truly perfect G-sampler algorithm for insertion only streams.

Input: A stream of updates $u_1, u_2, \ldots, u_m$, where each $u_i \in [n]$ represents a single update to a coordinate of the underlying vector f, a measure function G.

Initialize an instance of Sampler. ▷Algorithm 1
for each update $u_t \in [n]$ **do**
  Update Sampler with u_t.
Let s be the sampled output of Sampler and let c be the number of times s has appeared afterwards.
Let ζ be a parameter such that $G(x) - G(x-1) \leq \zeta$ for all $x \geq 1$.
return s with probability $\frac{G(c+1)-G(c)}{\zeta}$.

Theorem 3.1. *Let G be a function such that $0 \le G(x) - G(x-1) \le \zeta$ for all $x \ge 1$. Given a lower bound $\widehat{F_G}$ on $F_G = \sum_{i\in[n]} G(f_i)$, then there exists a truly perfect G sampler for an insertion-only stream that outputs **FAIL** with probability at most δ and uses $\mathcal{O}\left(\frac{\zeta m}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits of space. Further, the time to process each update is $\mathcal{O}(1)$ in expectation.*

Proof. The probability that s is the j^{th} particular instance of item i inside the stream is $\frac{1}{m}$. Since the number of instances of i appearing after j is $f_i - j$ then the probability that i is output is

$$\sum_{j=1}^{f_i} \frac{1}{m} \frac{G(f_i - j + 1) - G(f_i - j)}{\zeta} = \frac{G(f_i)}{\zeta m}.$$

We note that $G(f_i - j + 1) - G(f_i - j) \le \zeta$ for all $j \in [f_i]$, so returning s with probability $\frac{G(c+1)-G(c)}{\zeta}$ is valid.

Hence the probability that some index is returned by Algorithm 2 is $\sum_{i\in[n]} \frac{G(f_i)}{\zeta m} = \frac{F_G}{\zeta m}$, where $F_G = \sum G(f_i)$. Thus by repeating the sampler $\mathcal{O}\left(\frac{\zeta m}{F_G} \log \frac{1}{\delta}\right)$ times, the algorithm will output a sample s with probability at least $1 - \delta$. Although the algorithm does not actually have the value of F_G, given a lower bound $\widehat{F_G}$ on F_G, then it suffices to repeat the sampler $\mathcal{O}\left(\frac{\zeta m}{\widehat{F_G}} \log \frac{1}{\delta}\right)$ times. Moreover, the sample s will output each index $i \in [n]$ with probability $\frac{G(f_i)}{F_G}$. Each instance only requires $\mathcal{O}(\log n)$ bits to maintain the counter c, assuming $\log m = \mathcal{O}(\log n)$. Thus the total space used is $\mathcal{O}\left(\frac{\zeta m}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits of space.

We remark that the runtime of the algorithm can be optimized to constant time per update by storing a hash table containing a count and a list of offsets. Specifically, when item i is first sampled by some repetition of the algorithm, then we start counting the number of subsequent instances of i in the stream. If i is subsequently sampled by another independent instance of the reservoir sampling at some time t, then it suffices to store the value of the counter at the time t as an offset. This value does not change and can now be used to correctly recover the correct count of the number of instances of i after time t by subtracting this offset from the largest count. We can maintain a hash table with pointers to the head and tail of the list, so that when an item i is sampled, we can efficiently check whether that item is already being tracked by another repetition of the sampler. Hence the update time is $\mathcal{O}(1)$ worst-case once the hash bucket for i is determined and $\mathcal{O}(1)$ in expectation overall given the assignment of the bucket by the hash function. Note that by design of the offsets, we can build the correct counters at the end of the stream to determine the corresponding sampling probabilities. Finally, we observe that it is well-known how to optimize the runtime of the reservoir sampling over Algorithm 1 by using $\mathcal{O}(k \log n)$ total time to sample k items [Li94]. □

An interesting corollary of our results is to obtaining $s = \tilde{o}(n^{1/p})$ samples, rather than only a single sample. Our memory, like previous (non-truly perfect) samplers, does get multiplied by s, but our update time surprisingly remains $\mathcal{O}(1)$. By comparison, the only known previous perfect L_p sampler would require a prohibitive $s \cdot \text{poly}(n)$ update time [JW18b]. The reason our algorithm does not suffer from a multiplicative overhead in update time is because our data structure maintains a hash table containing a count and a list of offsets for each of the distinct items that is sampled during the stream. Thus, if our algorithm is required to output s samples, then our hash table now contains more items, but each stream update can only affect the counter and offset corresponding to the value of the update.

3.2 Applications in Data Streams

In this section, we show various applications of Algorithm 2 in the streaming model. The main barrier to applying Theorem 3.1 to any arbitrary measure function G is obtaining a "good" lower bound $\widehat{F_G}$ to $F_G = \sum_{i\in[n]} G(f_i)$.

3.2.1 Truly Perfect L_p Sampling on Insertion-Only Streams

We first consider truly perfect L_p sampling, where $G(x) = |x|^p$, for $p \geq 1$. Note that reservoir sampling is already a perfect L_1 sampler for $p = 1$ and it uses $\mathcal{O}(\log n)$ bits of space on a stream of length $m = \text{poly}(n)$. For $p \in (1, 2]$, we first require the following norm estimation algorithm on insertion-only streams.

We now introduce an algorithm that for *truly perfect* L_p sampling using the above framework. We first describe the case of $p = 2$ but before describing our perfect L_2 sampler, we first recall the MisraGries data structure for finding heavy-hitters.

Theorem 3.2. *[MG82] There exists a deterministic one-pass streaming algorithm* MisraGries *that uses $\mathcal{O}\left(\frac{1}{\epsilon}\log m\right)$ space on a stream of length m and outputs a list L of size $\frac{1}{2\epsilon}$ that includes all items i such that $f_i > 2\epsilon m$. Moreover, the algorithm returns an estimate $\widehat{f_i}$ for each $i \in L$ such that $f_i - \epsilon m \leq \widehat{f_i} \leq f_i$.*

Although we could obtain a perfect L_p sampler using any L_p estimation algorithm that succeeds with high probability, we can further remove the additive $\frac{1}{\text{poly}(n)}$ error of returning each coordinate $i \in [n]$ using the MisraGries data structure to obtain a *truly* perfect L_p sampler.

Theorem 3.3. *For a frequency vector f implicitly defined by an insertion-only stream, there exists an algorithm that returns each $i \in [n]$ with probability $\frac{f_i^p}{\sum_{j\in[n]} f_j^p}$, using space $\mathcal{O}\left(m^{1-p}\log n\right)$ for $p \in (0, 1]$ and $\mathcal{O}\left(n^{1-1/p}\log n\right)$ bits of space for $p \in [1, 2]$.*

We break down the proof of Theorem 3.3 into casework for $p \in [1, 2]$ and $p \in (0, 1]$.

Theorem 3.4. *For $p \in [1, 2]$ and a frequency vector f implicitly defined by an insertion-only stream, there exists an algorithm that returns each $i \in [n]$ with probability $\frac{f_i^p}{\sum_{j\in[n]} f_j^p}$, using $\mathcal{O}\left(n^{1-1/p}\log n\right)$ bits of space.*

Proof. By Theorem 3.2, using a single MisraGries data structure with $\mathcal{O}\left(n^{1-1/p}\log n\right)$ bits of space allows us to obtain a number Z such that

$$\|f\|_\infty \leq Z \leq \|f\|_\infty + \frac{m}{n^{1-1/p}}.$$

Note that for $p \in [1, 2]$, the function $x^p - (x-1)^p$ is maximized at $p = 2$, equal to $2x^{p-1}$, by the generalized binomial theorem. Since $f_i \leq \|f\|_\infty$, then we have $(f_i)^p - (f_i - 1)^p \leq 2\|f\|_\infty^{p-1}$ for any $i \in [n]$, so that $\zeta = 2Z^{p-1}$ induces a valid sampling procedure. Hence each instance outputs some index $i \in [n]$ with probability at least $\frac{F_p}{2Z^{p-1}m}$. If $\|f\|_\infty \geq \frac{m}{n^{1-1/p}}$, then we have $2Z \leq 4\|f\|_\infty \leq 4\|f\|_p$, so that

$$\frac{F_p}{2Z^{p-1}m} \geq \frac{F_p}{4L_p^{p-1}\cdot m} = \frac{L_p}{4F_1} \geq \frac{1}{4n^{1-1/p}}.$$

On the other hand, if $\|f\|_\infty \le \frac{m}{n^{1-1/p}}$, then we have $2Z \le \frac{4m}{n^{1-1/p}}$, so that

$$\frac{F_p}{2Z^{p-1}m} \ge \frac{F_p \cdot n^{(p-1)^2/p}}{4m^p} = \frac{F_p \cdot n^{(p-1)^2/p}}{4F_1^p} \ge \frac{F_p \cdot n^{(p-1)^2/p}}{4F_p \cdot n^{(p-1)}}$$

$$= \frac{1}{4n^{1-1/p}}.$$

Therefore, the probability that an instance outputs some index $i \in [n]$ is at least $\frac{1}{4n^{1-1/p}}$, and it suffices to use $\mathcal{O}\left(n^{1-1/p}\right)$ such instances, with total space $\mathcal{O}\left(n^{1-1/p}\log n\right)$ bits of space. By Theorem 3.2, conditioned on an index being returned by the algorithm, the probability that each coordinate $i \in [n]$ is output is $\frac{f_i^p}{F_p}$. □

Theorem 3.5. *For $p \in (0,1]$ and a frequency vector f implicitly defined by an insertion-only stream, there exists an algorithm that returns each $i \in [n]$ with probability $\frac{f_i^p}{\sum_{j\in[n]} f_j^p}$, using $\mathcal{O}\left(m^{1-p}\log n\right)$ bits of space.*

Proof. Note that for $p \in (0,1]$, we have $(f_i)^p - (f_i-1)^p \le 1$ for any $i \in [n]$, so that $\zeta = 1$ induces a valid sampling procedure. Hence each instance outputs some index $i \in [n]$ with probability at least $\frac{F_p}{m} \ge \frac{1}{m^{1-p}}$. Therefore, it suffices to use $\mathcal{O}\left(m^{1-p}\right)$ such instances, with total space $\mathcal{O}\left(m^{1-p}\log n\right)$ bits of space and conditioned on an index being returned by the algorithm, the probability that each coordinate $i \in [n]$ is output is $\frac{f_i^p}{F_p}$. □

Together, Theorem 3.4 and Theorem 3.5 give Theorem 3.3.

3.2.2 M-estimators on Insertion-Only Streams

We generalize the paradigm of Algorithm 2 to sampling from general statistical M-estimator distributions. Recall that for a given a measure function $G : \mathbb{R} \to \mathbb{R}^{\ge 0}$ such that $G(x) = G(-x)$, $G(0) = 0$ and G is non-decreasing in $|x|$, we define $F_G = \sum_{i=1}^n G(f_i)$. Then a truly perfect M-estimator sampler outputs index $i \in [n]$ with probability $\frac{G(f_i)}{F_G}$.

For the $L_1 - L_2$ estimator, we have $G(x) = 2\left(\sqrt{1+\frac{x^2}{2}} - 1\right)$ so that $G(x) - G(x-1) < 3$ for $x \ge 1$. For the Fair estimator, we have $G(x) = \tau|x| - \tau^2 \log\left(1 + \frac{|x|}{\tau}\right)$ for some constant $\tau > 0$ so that $G(x) - G(x-1) < \tau$ for $x \ge 1$. For the Huber measure function, we have $G(x) = \frac{x^2}{2\tau}$ for $|x| \le \tau$ and $G(x) = |x| - \frac{\tau}{2}$ otherwise, where $\tau > 0$ is some constant parameter.

Corollary 3.6. *There exist truly perfect G samplers for the insertion-only streaming model that succeed with probability at least $1-\delta$ and use $\mathcal{O}\left(\log n \log \frac{1}{\delta}\right)$ bits of space when G is the $L_1 - L_2$ estimator, the Fair estimator, or the Huber estimator.*

Proof. For the $L_1 - L_2$ estimator, we have $G(x) = 2\left(\sqrt{1+\frac{x^2}{2}} - 1\right)$ so that $G(x) - G(x-1) < 3$ for $x \ge 1$. Moreover, $G(x) > |x|$ so $F_G > m$. Hence by Theorem 3.1, there exists a perfect G sampler that uses $\mathcal{O}(\log n)$ bits of space when G is the $L_1 - L_2$ estimator.

For the Fair estimator, we have $G(x) = \tau|x| - \tau^2 \log\left(1 + \frac{|x|}{\tau}\right)$ for some constant $\tau > 0$ so that $G(x) - G(x-1) < \tau$ for $x \geq 1$. Since $G(x) > \tau|x|$ and thus $F_G > \tau m$, then by Theorem 3.1, there exists a perfect G sampler that uses $\mathcal{O}(\log n)$ bits of space when G is the Fair estimator.

For the Huber measure function, we have $G(x) = \frac{x^2}{2\tau}$ for $|x| \leq \tau$ and $G(x) = |x| - \frac{\tau}{2}$ otherwise, where $\tau > 0$ is some constant parameter. Hence, $G(x) - G(x-1) < 1$ and $G(x) > \frac{\tau}{2} \cdot m$, so there exists a perfect G sampler that uses $\mathcal{O}(\log n)$ bits of space when G is the Huber estimator by Theorem 3.1. □

3.2.3 Matrix Norms on Insertion-Only Streams

We now consider the case of sampling row $\mathbf{m}_i$ from a matrix $\mathbf{M} \in \mathbb{R}^{n \times d}$ with rows $\mathbf{m}_1, \ldots, \mathbf{m}_n \in \mathbb{R}^d$ with probability $\frac{G(\mathbf{m}_i)}{F_G}$ for some function G, where we define $F_G = \sum_{j \in [n]} G(\mathbf{m}_j)$. We consider a insertion-only stream in the sense that each update in the stream is a non-negative update to some coordinate of $\mathbf{M}$.

We generalize the approach of Algorithm 2 by first using reservoir sampling to sample an update to a coordinate c to a row r of $\mathbf{M}$. We then main a vector $\mathbf{v}$ that consists of all the updates to row r and choose to output r with probability $G(\mathbf{v} + e_c) - G(\mathbf{v})$, where e_c represents the elementary vector in $\mathbb{R}^d$ with a single 1 in coordinate c and zeros elsewhere.

Algorithm 3 Truly perfect G-sampler algorithm for vectors and matrices in insertion only streams.

Input: A stream of updates $u_1, u_2, \ldots, u_m$, where each $u_i \in [n] \times [d]$ represents a single update to a coordinate of a underlying matrix $\mathbf{M}$, a measure function G.

1: Initialize an instance of Sampler. ▷Algorithm 1
2: **for** each update $u_t \in [n] \times [d]$ **do**
3: Update Sampler.
4: Let r be the row and c be the column sampled by Sampler and let $\mathbf{v}$ be the vector induced by the subsequent updates to row r.
5: Let ζ be a parameter such that $G(\mathbf{x}) - G(\mathbf{x} - e_i) \leq \zeta$ for all $x \geq \left(\mathbb{R}^{\geq 0}\right)^d$, $i \in [d]$.
6: **return** r with probability $\frac{G(\mathbf{v}+e_c) - G(\mathbf{v})}{\zeta}$.

The correctness of Algorithm 3 follows from a similar proof to that of Theorem 3.1.

Theorem 3.7. *Fix any non-negative function $G : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ satisfying $G(\vec{0}) = 0$. Let ζ be a parameter such that $G(\mathbf{x}) - G(\mathbf{x} - e_i) \leq \zeta$ for all $\mathbf{x} \in \left(\mathbb{R}^{\geq 0}\right)^d$, $i \in [d]$. Given a lower bound $\widehat{F_G}$ on F_G, then there exists a truly perfect G sampler for an insertion-only stream that succeeds with probability at least $1 - \delta$ and uses $\mathcal{O}\left(\frac{\zeta dm}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits of space.*

Proof. The probability that the update to (r, c) is the j^{th} particular update to row r inside the stream is $\frac{1}{m}$. Let $\mathbf{x}_j$ be the sum of the updates to row r after the j^{th} update and let e_{c_j} be the coordinate of row r incremented in the j^{th} update, so that the probability that i is output is

$$\sum_j \frac{1}{m} \frac{G(\mathbf{x}_j + e_{c_j}) - G(\mathbf{x}_j)}{\zeta} = \sum_j \frac{1}{m} \frac{G(\mathbf{x}_{j-1}) - G(\mathbf{x}_j)}{\zeta}$$

$$= \frac{G(\mathbf{m}_i)}{\zeta m},$$

where the final equality results from the observations that $\mathbf{x}_0 = \mathbf{m}_r$ and that $G(\mathbf{0}) = 0$, since $\mathbf{v}$ must be the all zeros vector after the last update to row r. Thus conditioned on some row being output, the algorithm outputs each row $i \in [n]$ with probability $\frac{G(f_i)}{F_G}$. We again note that $G(\mathbf{x}_j + e_{c_j}) - G(\mathbf{x}_j) \leq \zeta$ for all $\mathbf{x} \in \left(\mathbb{R}^{\geq 0}\right)^d$, $i \in [d]$, so returning r with probability $\frac{G(\mathbf{x}_j + e_{c_j}) - G(\mathbf{x}_j)}{\zeta}$ is well-defined.

Therefore, the probability that some row is returned by Algorithm 3 is $\sum_{i \in [n]} \frac{G(\mathbf{m}_i)}{\zeta m} = \frac{F_G}{\zeta m}$, where $F_G = \sum G(\mathbf{m}_i)$. By repeating the sampler $\mathcal{O}\left(\frac{\zeta m}{F_G} \log \frac{1}{\delta}\right)$ times, the algorithm successfully outputs a sample s with probability at least $1 - \delta$. We again note that although the algorithm does not know the value of F_G, it suffices to repeat the sampler $\mathcal{O}\left(\frac{\zeta m}{\widehat{F_G}} \log \frac{1}{\delta}\right)$ times for some lower bound $\widehat{F_G}$ on F_G. Each instance only requires $\mathcal{O}(d \log n)$ bits to maintain the vector $\mathbf{v}$, assuming $\log m = \mathcal{O}(\log n)$ and each update is bounded by $\text{poly}(n)$, which can be expressed using $\mathcal{O}(\log n)$ bits. Thus the total space used is $\mathcal{O}\left(\frac{\zeta dm}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits of space. □

For example, when $G(\mathbf{x}) = \sum_{i \in [d]} |x_i|$, then F_G is the $L_{1,1}$ norm. Then $F_G = m$, so that by Theorem 3.7, so we can sample a row $\mathbf{m}_i$ with probability proportional to its L_1 norm, using $\mathcal{O}\left(d \log n \log \frac{1}{\delta}\right)$ bits of space. We can also apply Theorem 3.7 when $G(\mathbf{x}) = \sqrt{\sum_{i \in [d]} x_i^2}$ is the L_2 norm of each row, so that F_G is the $L_{1,2}$ norm crucially used in many adaptive sampling techniques (see [MRWZ20] and references therein).

Finally, we show in Appendix D that our framework can be extended to strict turnstile streams, i.e., Theorem 1.5.

4 Applications in Sliding Windows

In this section, we give additional applications of our framework to truly perfect samplers on sliding windows. Recall that in the sliding window model, updates $u_1, \ldots, u_m$ to an underlying vector $f \in \mathbb{R}^n$ arrive sequentially as a data stream and the underlying vector f is determined by the most recent W updates $u_{m-W+1}, \ldots, u_m$, where $W > 0$ is the predetermined window size parameter. We assume that m and W are polynomially bounded in n, i.e., $\mathcal{O}(\log m) = \mathcal{O}(\log W) = \mathcal{O}(\log n)$. For each update u_k, if $k < m - W + 1$, we say u_k is *expired*. Otherwise if $k \geq m - W + 1$, we say u_k is *active*.

4.1 M-estimators on Sliding Windows

In this section, we consider a general paradigm for sampling from general statistical M-estimator distributions. Recall that for a given a measure function $G : \mathbb{R} \to \mathbb{R}^{\geq 0}$ such that $G(x) = G(-x)$, $G(0) = 0$ and G is non-decreasing in $|x|$, we define $F_G = \sum_{i=1}^n G(f_i)$ so that F_G is also implicitly defined by only the most recent W updates. Then a truly perfect M-estimator sampler outputs index $i \in [n]$ with probability exactly $\frac{G(f_i)}{F_G}$.

The key argument in Theorem 1.4 was that if c instances of the sample s appeared after the initial sample, then s is output with probability proportional to $c^p - (c-1)^p$. By a telescoping argument, each index i is sampled with probability proportional to $\sum_{c=1}^{f_i} c^p - (c-1)^p = f_i^p$. Since G is non-decreasing in $|x|$, we can generalize to sampling each item with probability proportional

to $G(c)-G(c-1)$, rather than $c^p-(c-1)^p$. This approach can be simulated in the sliding window model by checking whether s is an active element.

Algorithm 4 Truly perfect M-estimator sampler algorithm for the sliding window model on insertion only streams.

Input: A stream of updates $u_1, u_2, \ldots, u_m$, where each $u_i \in [n]$ represents a single update to a coordinate of the underlying vector f, a measure function G, and a size W for the sliding window.

1: Initialize instances of Sampler every W updates and keep the two most recent instances. ▷Algorithm 1
2: **for** each update $u_t \in [n]$ with $t \in [m]$ **do**
3: Update each Sampler.
4: Let s be the sampled output of a Sampler and let c be the number of times s has appeared afterwards.
5: Let ζ be a parameter such that $G(x)-G(x-1) \le \zeta$ for all $x \ge 1$.
6: **if** s was sampled within the last W updates **then**
7: **return** s with probability $\frac{G(c+1)-G(c)}{\zeta}$.

Theorem 4.1. *Let G be a function such that $G(x)-G(x-1) \le \zeta$ for all $x \ge 1$. Then there exists a truly perfect G sampler for the sliding window model that succeeds with probability at least $1-\delta$ and uses $\mathcal{O}\left(\frac{\zeta W}{F_G}\log n \log\frac{1}{\delta}\right)$ bits of space.*

Proof. As in Theorem 1.4, Sampler contains not only active elements, but an additional number of expired elements. If s is an active element, then the probability that s is the j^{th} particular instance of item i inside the window $\frac{1}{W}$. Since the number of instances of i appearing after j is $f_i - j + 1$ then the probability that i is output is

$$\sum_{j=1}^{f_i} \frac{1}{W}\frac{G(f_i-j+1)-G(f_i-j)}{\zeta} = \frac{G(f_i)}{\zeta W}.$$

Again we note that $G(f_i-j+1)-G(f_i-j) \le \zeta$ for all $j \in [f_i]$, so returning s with probability $\frac{G(c)-G(c-1)}{\zeta}$ is valid.

Hence if s is an active element, the probability that s is output is at least $\frac{F_G}{\zeta W}$, where $F_G = \sum G(f_i)$. Thus by repeating the sampler $\mathcal{O}\left(\frac{\zeta W}{F_G}\log\frac{1}{\delta}\right)$ times, the algorithm will output a sample s with probability at least $1-\delta$. Moreover, the sample s will equal each index $i \in [n]$ with probability $\frac{G(f_i)}{F_G}$.

Note that a single instance requires $\mathcal{O}(\log n)$ bits to maintain the counter c, assuming $\log W = \mathcal{O}(\log n)$. Thus the total space used is $\mathcal{O}\left(\frac{\zeta W}{F_G}\log n \log\frac{1}{\delta}\right)$ bits of space. □

Using Theorem 4.1 and the properties of the M-estimators defined in Corollary 3.6, we have:

Corollary 4.2. *There exist truly perfect G samplers for the insertion-only sliding window model that succeed with probability at least $1-\delta$ and use $\mathcal{O}\left(\log n \log\frac{1}{\delta}\right)$ bits of space when G is the L_1-L_2 estimator, the Fair estimator, or the Huber estimator.*

We also give truly perfect L_p samplers on sliding windows. Specifically we prove Theorem 1.4, the details of which can be found in Appendix A.

5 F_0 Sampling

In this section, we give truly perfect F_0 samplers in the insertion-only streaming model. In the F_0 sampling problem, the goal is to sample a coordinate $i \in [n]$ from a frequency vector of length n such that $\mathbf{Pr}[i = j] = 0$ if $f_j = 0$ and $\mathbf{Pr}[i = j] = \frac{1}{F_0}$ otherwise, where $F_0 = |\{i \in [n],\ f_i \neq 0\}|$. We cannot immediately apply the framework of Algorithm 2 for F_0 sampling without trivializing the space complexity, due to the fact that F_0 can be substantially smaller than m.

We first remark that in the random oracle model, where an algorithm is given oracle access to a random hash function $h : [n] \to [0, 1]$, the well-known algorithm that outputs the nonzero coordinate $i \in [n]$ of f_i that minimizes $h(i)$ is truly perfect F_0 sampler, since each of the $|F_0|$ has probability $\frac{1}{|F_0|}$ of obtaining the minimal hash value for a random hash function. The random oracle model allows the generation of, and repeated access to any number of random bits, and in particular it allows for repeated access to the same $\Omega(n)$ random bits without charging the algorithm for storing the random bits.

Remark 5.1. *In the random oracle model, there exists a truly perfect F_0 sampler on insertion-only streams that uses $\mathcal{O}(\log n)$ bits of space and constant update time.*

Note that storing $\Omega(n)$ random bits without the random oracle assumption is not interesting in the streaming model, as it corresponds to storing the entire input, up to logarithmic factors. We now give a truly perfect F_0 sampler on insertion-only streams without the assumption of the random oracle model. We store the first $\sqrt{n}$ distinct items and in parallel sample a set S of $\mathcal{O}(\sqrt{n})$ random items from the universe. We maintain the subset U of S that arrive in the stream. Now since we store the first $\sqrt{n}$ distinct items, we know whether $F_0 \leq \sqrt{n}$ or $F_0 > \sqrt{n}$. If $F_0 \leq \sqrt{n}$ then our algorithm has collected all items in the stream and can simply output an item uniformly at random. Otherwise if F_0 is larger than $\sqrt{n}$ at the end of the stream, then since S has size $\mathcal{O}(\sqrt{n})$ and was generated at random, we have constant probability that some element of S has arrived in the stream. Our algorithm can then output a random element of U that has appeared in the stream. We give our algorithm in full in Algorithm 5.

Algorithm 5 Truly perfect F_0 sampler for insertion only streams.

Input: A stream of updates $u_1, u_2, \ldots, u_m$, where each $u_i \in [n]$ represents a single update to a coordinate of a underlying vector f.

Let S be a random subset of $[n]$ of size $2\sqrt{n}$.
Let T be the first unique $\sqrt{n}$ coordinate updates to f in the stream.
Let U be the subset of S that appears in the stream.
if $|T| < \sqrt{n}$ **then**
    return a random element of T
else if $|U| > 0$ **then**
    return a random element of U
else
    return FAIL

Theorem 5.2. *Given $\delta \in (0,1)$, there exists a truly perfect F_0 sampler on insertion-only streams that uses $\mathcal{O}\left(\sqrt{n}\log n \log\frac{1}{\delta}\right)$ bits of space and constant update time and succeeds with probability at least $1-\delta$. Moreover, the algorithm reports f_i along with the sampled index $i \in [n]$.*

Proof. Consider Algorithm 5. If $F_0 < \sqrt{n}$, then all nonzero coordinates of f will be stored in T in the insertion-only streaming model and so a random element of T is a truly perfect F_0 sample. Otherwise if $F_0 \geq \sqrt{n}$ and the algorithm does not return **FAIL**, then it must have output a random item in U, which is a subset of S. Since S is a subset of n chosen uniformly at random, then a random element of S is a truly perfect F_0 sample.

It remains to analyze the probability that Algorithm 5 returns **FAIL** when $F_0 \geq \sqrt{n}$, since it will never fail when $F_0 < \sqrt{n}$. Our algorithm will only fail if $|U| = 0$, so that none of the $2\sqrt{n}$ random items in S appeared in the stream. This can only occur with probability at most $\left(1-\frac{2\sqrt{n}}{n}\right)^{\sqrt{n}}$, which is certainly at most $\frac{1}{e}$ for sufficiently large n. Hence by repeating $\mathcal{O}\left(\log\frac{1}{\delta}\right)$ times, the algorithm has probability at least $1-\delta$ of success. Then the space required is $\mathcal{O}\left(\sqrt{n}\log n\log\frac{1}{\delta}\right)$.

Finally, note that the algorithm can track the frequency of each element in U and T, so the algorithm can also report the frequency f_i corresponding to the sampled index i. □

By modifying T to be the last unique $\sqrt{n}$ coordinate updates to f in the stream and storing timestamps for each element in U, Algorithm 5 extends naturally to the sliding window model.

Corollary 5.3. *Given $\delta \in (0,1)$, there exists a truly perfect F_0 sampler in the sliding window model that uses $\mathcal{O}\left(\sqrt{n}\log n\log\frac{1}{\delta}\right)$ bits of space and constant update time and succeeds with probability at least $1-\delta$.*

Recall that for the Tukey measure function, we have $G(x) = \frac{\tau^2}{6}\left(1-\left(1-\frac{x^2}{\tau^2}\right)^3\right)$ for $|x| \leq \tau$ and $G(x) = \frac{\tau^2}{6}$ otherwise, where $\tau > 0$ is some constant. We can now use our F_0 sampler of choice, say L0Sampler, to obtain a truly perfect sampler for the Tukey measure function by a similar approach to Algorithm 2. Each instance will use a subroutine of L0Sampler rather than Sampler. Now if c is the number of instances of the index output by L0Sampler within the window, then we accept c with probability $\frac{G(c)}{G(\tau)}$.

Theorem 5.4. *Given $\delta \in (0,1)$, there exists a truly perfect G sampler for the insertion-only streaming model that succeeds with probability at least $1-\delta$ when G is the Tukey estimator. The algorithm uses $\mathcal{O}\left(\log n\log\frac{1}{\delta}\right)$ bits of space in the random oracle model and $\mathcal{O}\left(\sqrt{n}\log n\log\frac{1}{\delta}\right)$ otherwise.*

Proof. Note that only the indices that appear in the stream can be output. The probability that any index $i \in [n]$ that ap pears in the stream is output is $\frac{1}{F_0}\cdot\frac{G(f_i)}{G(\tau)}$. Then a single instance of the algorithm returns an output with probability $\frac{F_G}{F_0\cdot G(\tau)} \geq \frac{G(1)}{G(\tau)}$. Hence repeating $\mathcal{O}\left(\log\frac{1}{\delta}\right)$ times outputs an instance with probability at least $1-\delta$. Since L0Sampler requires $\mathcal{O}(\log n)$ space in the random oracle model, then the total space used is $\mathcal{O}\left(\log n\log\frac{1}{\delta}\right)$ bits of space. Otherwise by Theorem 5.2, L0Sampler requires $\mathcal{O}\left(\sqrt{n}\log n\right)$ space so that the total space used is $\mathcal{O}\left(\sqrt{n}\log n\log\frac{1}{\delta}\right)$. □

We also obtain a truly perfect G sampler for the insertion-only sliding window model when G is the Tukey estimator by a similar approach to Algorithm 4. We again use a truly perfect F_0 sampling algorithm L0Sampler of choice and, if c is the number of instances of the index s output by L0Sampler within the window, then we accept s with probability $\frac{G(c)}{G(\tau)}$.

Theorem 5.5. *Given $\delta \in (0,1)$, there exists a truly perfect G sampler for the sliding window model that succeeds with probability at least $1-\delta$ when G is the Tukey estimator. The algorithm uses $\mathcal{O}\left(\log n \log \frac{1}{\delta}\right)$ bits of space in the random oracle model and $\mathcal{O}\left(\sqrt{n} \log n \log \frac{1}{\delta}\right)$ otherwise.*

Finally, we show in Appendix D that the same guarantees of Theorem 5.2 can be extended to strict turnstile streams.

6 Conclusion

Our work shows that surprisingly, truly perfect samplers exist in the insertion-only model with a sublinear amount of memory for a large class of loss functions, a large class of objects (vectors, matrices), and several different models (data stream and sliding window). Establishing better upper bounds, or proving lower bounds is a very intriguing open question. In particular, the usual communication complexity lower bound arguments do not seem to apply, and already for our turnstile lower bound we needed to use fine-grained properties of the equality function.

Acknowledgements

This work was supported by National Science Foundation (NSF) Grant No. CCF-1815840, National Institute of Health (NIH) grant 5R01 HG 10798-2, and a Simons Investigator Award.

References

[ACSS21] 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. 4

[ADK09] Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX and 13th International Workshop, RANDOM. Proceedings*, pages 15–28, 2009. 2

[AKO11] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 363–372, 2011. 2, 3, 4, 5, 9

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. 8

[AMYZ19] Dmitrii Avdiukhin, Slobodan Mitrovic, Grigory Yaroslavtsev, and Samson Zhou. Adversarially robust submodular maximization under knapsack constraints. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 148–156, 2019. 4

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM*

SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pages 1–16, 2002. 4

[BCK+14] Joshua Brody, Amit Chakrabarti, Ranganath Kondapally, David P Woodruff, and Grigory Yaroslavtsev. Certifying equality with limited interaction. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, 2014. 9, 10

[BDM+20] Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 517–528, 2020. 4

[BEJWY20] Omri Ben-Eliezer, Rajesh Jayaram, David P Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 63–80, 2020. 4

[BEY20] 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*, pages 49–62, 2020. 4

[BGL+18] Vladimir Braverman, Elena Grigorescu, Harry Lang, David P. Woodruff, and Samson Zhou. Nearly optimal distinct elements and heavy hitters on sliding windows. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 7:1–7:22, 2018. 4, 32

[BHM+21] 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. 4

[BLLM15] Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering on sliding windows in polylogarithmic space. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, pages 350–364, 2015. 4

[BLLM16] Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering problems on sliding windows. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1374–1390, 2016. 4

[BO07] Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Proceedings*, pages 283–293, 2007. 4, 27, 28

[BOZ12] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.*, 78(1):260–272, 2012. 2, 4, 6

[CCD11] Edith Cohen, Graham Cormode, and Nick G. Duffield. Structure-aware sampling: Flexible and accurate summarization. *Proc. VLDB Endow.*, 4(11):819–830, 2011. 2

[CCD12] Edith Cohen, Graham Cormode, and Nick G. Duffield. Don't let the negatives bring you down: sampling from streams of signed updates. In *ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS*, pages 343–354, 2012. 2

[CDK+11] Edith Cohen, Nick G. Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Efficient stream sampling for variance-optimal estimation of subset sums. *SIAM J. Comput.*, 40(5):1402–1431, 2011. 2

[CDK+14] Edith Cohen, Nick G. Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Algorithms and estimators for summarization of unaggregated data streams. *J. Comput. Syst. Sci.*, 80(7):1214–1244, 2014. 2

[CG08] Graham Cormode and Minos N. Garofalakis. Streaming in a connected world: querying and tracking distributed data streams. In *EDBT 2008, 11th International Conference on Extending Database Technology, Proceedings*, page 745, 2008. 4

[CG19] Edith Cohen and Ofir Geri. Sampling sketches for concave sublinear functions of frequencies. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, pages 1361–1371, 2019. 2, 3

[CJ19] Graham Cormode and Hossein Jowhari. L p samplers and their applications: A survey. *ACM Computing Surveys (CSUR)*, 52(1):1–31, 2019. 3

[CJLW20] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. An improved analysis of the quadtree for high dimensional emd, 2020. 5

[CM05] Graham Cormode and S. Muthukrishnan. What's new: finding significant differences in network data streams. *IEEE/ACM Transactions on Networking*, 13(6):1219–1232, 2005. 4

[CMR05] Graham Cormode, S. Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 25–36. ACM, 2005. 2

[CMYZ10] Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang. Optimal sampling from distributed streams. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 77–86, 2010. 2

[CMYZ12] Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang. Continuous sampling from distributed streams. *J. ACM*, 59(2):10:1–10:25, 2012. 2

[CNZ16] Jiecao Chen, Huy L. Nguyen, and Qin Zhang. Submodular maximization over sliding windows. *CoRR*, abs/1611.00129, 2016. 4

[Coh18] Edith Cohen. Stream sampling framework and application for frequency cap statistics. *ACM Trans. Algorithms*, 14(4):52:1–52:40, 2018. 2

[Cor13] Graham Cormode. The continuous distributed monitoring model. *SIGMOD Record*, 42(1):5–14, 2013. 4

[CPW20] Edith Cohen, Rasmus Pagh, and David P. Woodruff. WOR and p's: Sketches for ℓ_p-sampling without replacement. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2020. 2

[Dat16] Deepesh Data. Secure computation of randomized functions. In *IEEE International Symposium on Information Theory, ISIT*, pages 3053–3057, 2016. 3

[DGIM02] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. 4

[DM07] Mayur Datar and Rajeev Motwani. The sliding-window computation model and results. In *Data Streams - Models and Algorithms*, pages 149–167. Springer, 2007. 4

[DRVW06] Amit Deshpande, Luis Rademacher, Santosh S. Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. *Theory of Computing*, 2(12):225–247, 2006. 2

[DV06] Amit Deshpande and Santosh S. Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop, APPROX and 10th International Workshop, RANDOM. Proceedings*, pages 292–303, 2006. 2

[DV07] Amit Deshpande and Kasturi R. Varadarajan. Sampling-based dimension reduction for subspace approximation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 641–650, 2007. 2

[ELVZ17] Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. Submodular optimization over sliding windows. In *Proceedings of the 26th International Conference on World Wide Web, WWW*, pages 421–430, 2017. 4

[EV03] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003. 2

[FKV04] Alan M. Frieze, Ravi Kannan, and Santosh S. Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *J. ACM*, 51(6):1025–1041, 2004. 2

[Gan08] Sumit Ganguly. Data stream algorithms via expander graphs. In *Algorithms and Computation, 19th International Symposium, ISAAC. Proceedings*, pages 52–63, 2008. 40

[GKM18] Parikshit Gopalan, Daniel M. Kane, and Raghu Meka. Pseudorandomness via the discrete fourier transform. *SIAM J. Comput.*, 47(6):2451–2487, 2018. 33

[GKMS01] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin Strauss. Quicksand: Quick summary and analysis of network data, 2001. Technical report. 2

[GLH08] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. Maintaining bounded-size sample synopses of evolving datasets. *VLDB J.*, 17(2):173–202, 2008. 2

[GM98] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 331–342, 1998. 2

[GM08] Sumit Ganguly and Anirban Majumder. Deterministic k-set structure. *Inf. Process. Lett.*, 109(1):27–31, 2008. 40

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009. 40

[Haa16] Peter J. Haas. Data-stream sampling: Basic techniques and results. In *Data Stream Management - Processing High-Speed Data Streams*, Data-Centric Systems and Applications, pages 13–44. Springer, 2016. 2

[Hal81] Peter Hall. On the rate of convergence to a stable law. *Journal of the London Mathematical Society*, 2(1):179–192, 1981. 35

[HKM+20] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. In *Advances in Neural Information Processing Systems*, 2020. 4

[HNG+07] Ling Huang, XuanLong Nguyen, Minos N. Garofalakis, Joseph M. Hellerstein, Michael I. Jordan, Anthony D. Joseph, and Nina Taft. Communication-efficient online detection of network-wide anomalies. In *INFOCOM. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pages 134–142, 2007. 2

[HNSS96] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. Selectivity and cost estimation for joins based on random sampling. *J. Comput. Syst. Sci.*, 52(3):550–569, 1996. 2

[HPGS16] Xiaowei Hu, LA Prashanth, András György, and Csaba Szepesvari. (bandit) convex optimization with biased noisy gradient oracles. In *Artificial Intelligence and Statistics*, pages 819–828. PMLR, 2016. 3

[HS92] Peter J. Haas and Arun N. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 341–350, 1992. 2

[IRU+19] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Ion Stoica, Raman Arora, et al. Communication-efficient distributed sgd with sketching. In *Advances in Neural Information Processing Systems*, pages 13144–13154, 2019. 3

[JST11] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 49–58, 2011. 2, 3, 4, 5, 9

[JSTW19] Rajesh Jayaram, Gokarna Sharma, Srikanta Tirthapura, and David P. Woodruff. Weighted reservoir sampling from distributed streams. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS*, pages 218–235, 2019. 2, 5

[JW18a] Rajesh Jayaram and David P. Woodruff. Data streams with bounded deletions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database System*, pages 341–354, 2018. 5

[JW18b] Rajesh Jayaram and David P. Woodruff. Perfect L_p sampling in a data stream. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 544–555, 2018. 2, 3, 4, 5, 7, 9, 12, 30, 31, 33

[JZ13] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *27th Annual Conference on Neural Information Processing Systems*, pages 315–323, 2013. 3

[Li94] Kim-Hung Li. Reservoir-sampling algorithms of time complexity o(n (1+ log (n/n))). *ACM Transactions on Mathematical Software (TOMS)*, 20(4):481–493, 1994. 12

[LN95] Richard J. Lipton and Jeffrey F. Naughton. Query size estimation by adaptive sampling. *J. Comput. Syst. Sci.*, 51(1):18–25, 1995. 2

[LNS90] Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–11, 1990. 2

[LT06a] Lap-Kei Lee and H. F. Ting. Maintaining significant stream statistics over sliding windows. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 724–732, 2006. 4

[LT06b] Lap-Kei Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 290–297, 2006. 4

[MBN+17] Slobodan Mitrovic, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Volkan Cevher. Streaming robust submodular maximization: A partitioned thresholding approach. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 4557–4566, 2017. 4

[MCS+06] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference, IMC*, pages 165–176, 2006. 2

[MG82] Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982. 13

[MM12] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. *PVLDB*, 5(12):1699, 2012. 4

[MRWZ20] Sepideh Mahabadi, Ilya P. Razenshteyn, David P. Woodruff, and Samson Zhou. Non-adaptive adaptive sampling on turnstile streams. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1251–1264, 2020. 2, 6, 16

[MW10] Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error L_p-sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1143–1160, 2010. 2, 3, 4, 9

[Nag06] Haikady N Nagaraja. Order statistics from independent exponential random variables and the sum of the top order statistics. In *Advances in Distribution Theory, Order Statistics, and Inference*, pages 173–185. Springer, 2006. 30

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. 33

[NSW16] Deanna Needell, Nathan Srebro, and Rachel Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Math. Program.*, 155(1-2):549–573, 2016. 3

[OMM+14] Miles Osborne, Sean Moran, Richard McCreadie, Alexander Von Lunen, Martin Sykora, Elizabeth Cano, Neil Ireson, Craig MacDonald, Iadh Ounis, Yulan He, Tom Jackson, Fabio Ciravegna, and Ann O'Brien. Real-time detection, tracking and monitoring of automatically discovered events in social media. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014. 4

[PGD15] Odysseas Papapetrou, Minos N. Garofalakis, and Antonios Deligiannakis. Sketching distributed sliding-window data streams. *VLDB J.*, 24(3):345–368, 2015. 4

[TLJ10] Marina Thottan, Guanglei Liu, and Chuanyi Ji. Anomaly detection approaches for communication networks. In *Algorithms for Next Generation Networks*, Computer Communications and Networks, pages 239–261. Springer, 2010. 2

[TW11] Srikanta Tirthapura and David P. Woodruff. Optimal random sampling from distributed streams revisited. In *Distributed Computing - 25th International Symposium, DISC. Proceedings*, volume 6950, pages 283–297, 2011. 2

[Vit85] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. 2

[WLL+16] Zhewei Wei, Xuancheng Liu, Feifei Li, Shuo Shang, Xiaoyong Du, and Ji-Rong Wen. Matrix sketching over sliding windows. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference*, pages 1465–1480, 2016. 4

[WZ16] David P. Woodruff and Peilin Zhong. Distributed low rank approximation of implicit functions of a matrix. In *32nd IEEE International Conference on Data Engineering, ICDE*, pages 847–858, 2016. 2

[WZ20] David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. *CoRR*, abs/2011.07471, 2020. 4

[ZZ15] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, volume 37, pages 1–9, 2015. 3

A Proofs missing from Section 4

We first describe our truly perfect L_p samplers for sliding windows.

Smooth Histograms. Although the smooth histogram framework cannot be used for L_p sampling, we require the approximation of a number of parameters for which the smooth histogram framework can be used. We thus provide the necessary background. Braverman and Ostrovsky introduced the smooth histogram framework to give algorithms for solve a large number of problems in the sliding window model, such as L_p norm estimation, longest increasing subsequence, geometric mean estimation, or other weakly additive functions [BO07]. The smooth histogram framework is defined using the following notion of smooth functions, where the notation $B \subseteq_s A$ denotes that a stream B arrives at the end of stream A, so that B is a substream of A.

Definition A.1 (Smooth function)**.** *[BO07] A function f is (α, β)-smooth if for any stream A if*

(1) $0 \leq f(A) \leq \text{poly}(W)$

(2) $f(B) \leq f(A)$ *for* $B \subseteq_s A$

(3) For any $0 < \epsilon < 1$, there exists $0 < \beta \leq \alpha < 1$ such that if $B \subseteq_s A$ and $(1-\beta)f(A) \leq f(B)$, then $(1-\alpha)f(A \cup C) \leq f(B \cup C)$ any adjacent substream C.

Intuitively, a smooth function requires that once a suffix of a data stream becomes a $(1 \pm \beta)$-approximation for a smooth function, then it remains a $(1 \pm \alpha)$-approximation of the data stream, regardless of the subsequent updates that arrive in the stream.

The smooth histogram data structure maintains a number of timestamps throughout the data stream, along with a streaming algorithm that stores a sketch of all elements seen in the stream beginning at each timestamp. The smooth histogram maintains the invariant that at most three streaming algorithms output values that are within $(1-\beta)$ of each other, since any two of the sketches would always output values that are within $(1-\alpha)$ afterwards. Thus for polynomially bounded monotonic functions, only a logarithmic number of timestamps need to be stored. See Figure 1 for intuition on how the sliding window is sandwiched between two timestamps of the smooth histogram.

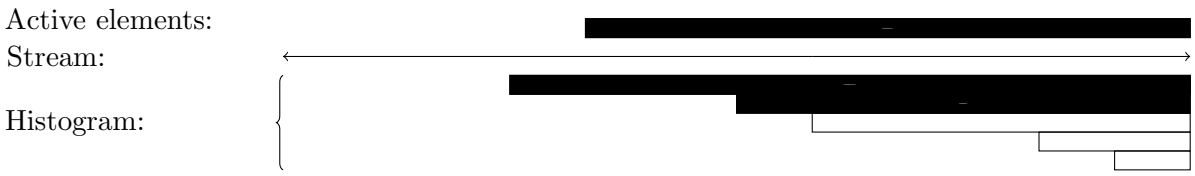

Fig. 1: Histogram paradigm. Note the first two algorithms sandwich the active elements.

The smooth histogram has the following properties:

Definition A.2 (Smooth Histogram)**.** *[BO07] Let g be a function that maintains a $(1+\epsilon)$-approximation of an (α,β)-smooth function f that takes as input a starting index and ending index in the data stream. The approximate smooth histogram is a structure that consists of an increasing set of indices $X_N = \{x_1,\ldots,x_s = N\}$ and s instances of an algorithm Λ, namely $\Lambda_1,\ldots,\Lambda_s$ with the following properties:*

(1) x_1 corresponds to either the beginning of the data stream, or an expired point.

(2) x_2 corresponds to an active point.

(3) For all $i < s$, one of the following holds:

(a) $x_{i+1} = x_i + 1$ and $g(x_{i+1},N) < \left(1-\frac{\beta}{2}\right) g(x_i,N)$.

(b) $(1-\alpha)g_(x_i,N) \le g(x_{i+1},N)$ and if $i+2 \le s$ then $g(x_{i+2},N) < \left(1-\frac{\beta}{2}\right) g(x_i,N)$.

(4) $\Lambda_i = \Lambda(x_i,N)$ maintains $g(x_i,N)$.

We define an augmented histogram to be a smooth histogram with additional auxiliary algorithms corresponding to each timestamp.

Definition A.3 (Augmented Histogram)**.** *An* augmented histogram *on a data stream $u_1,\ldots,u_t$ is a data structure $\mathcal{H}$ that consists of algorithms $\{\mathcal{A}_1^{(i)},\ldots,\mathcal{A}_s^{(i)}\}_i$, each with a corresponding time $t_1,\ldots,t_s$. The input to each algorithm $\mathcal{A}_j^{(i)}$ is the sequence of updates $u_{t_j}, u_{t_j+1},\ldots,u_t$, so that the input to $\mathcal{A}_j^{(i)}$ is the same as the input to $\mathcal{A}_j^{(k)}$.*

The following theorem shows the smoothness of $F_p = \sum_{i=1}^n f_i^p$.

Theorem A.4. *[BO07] For $p \ge 1$, F_p is $\left(\epsilon, \frac{\epsilon^p}{p^p}\right)$-smooth. For $p < 1$, F_p is (ϵ,ϵ)-smooth.*

We run several instances of the same algorithm. In each instance, we first use reservoir sampling to sample each item with probability $\frac{1}{2W}$. If the stream ends and the sampled item is outside the sliding window, then nothing is output by the algorithm and we move onto the next instance of the algorithm. When an item s in the sliding window is selected by the reservoir sampling procedure, we keep a counter c for how many times the item appears afterward. We also use a F_2 estimation algorithm to find a 2-approximation F of $\sqrt{F_2}$. We then output s with probability $\frac{c^2-(c-1)^2}{F}$. Otherwise, if we choose not to output s, we again move onto the next instance of the algorithm.

It can be shown through a straightforward telescoping argument that the probability of outputting each $i \in [n]$ is $\frac{f_i^2}{2FW}$. Since $W = F_1$ is greater than F_2 by at most a factor of $\sqrt{W}$, the probability that a single instance of the algorithm outputs some index is at least $\frac{1}{2\sqrt{W}}$. Hence running $\mathcal{O}\left(\sqrt{W}\right)$ instances of the algorithm is a perfect L_2 sampler.

We recall the following subroutine for norm estimation in the sliding window model.

Theorem A.5. *[BO07] There exists a sliding window algorithm* Estimate *that outputs an estimate f such that $f \le L_p \le 2f$, with probability $1 - \frac{1}{\operatorname{poly} n}$. The algorithm uses $\mathcal{O}\left(\log^2 n\right)$ space.*

Algorithm 6 Perfect L_p sampler for the sliding window model on insertion only streams and $p > 1$.

Input: A stream of updates $u_1, u_2, \ldots, u_t$, where each $u_i \in [n]$ represents a single update to a coordinate of the underlying vector f, and a size W for the sliding window.

Use an augmented histogram $\mathcal{H}$, where $\mathcal{A}_i^{(1)}$ is an instance of Estimate and $\mathcal{A}_i^{(2)}$ is an instance of Sampler.
for each update u_r **do**
  Let $\mathcal{H}$ consist of algorithms $\mathcal{A}_1^{(j)}, \ldots, \mathcal{A}_s^{(j)}$, where $j \in \{1, 2\}$.
  Initialize $\mathcal{A}_{s+1}^{(1)}$ as an instance of Estimate starting with u_r.
  Initialize $\mathcal{A}_{s+1}^{(2)}$ as an instance of Sampler starting with u_r. ▷Algorithm 1
  Set $t_{s+1} = r$.
  for each $1 \le i \le s$ **do**
    Update each $\mathcal{A}_i^{(1)}$, $\mathcal{A}_i^{(2)}$ with u_r.
  for each $2 \le i \le s-1$ **do**
    Let N_i be the output of $\mathcal{A}_i^{(1)}$.
    if $N_{i-1} \le 2N_{i+1}$ **then**
      Delete $\mathcal{A}_i^{(1)}$, $\mathcal{A}_i^{(2)}$, and t_i from $\mathcal{H}$.
      Reindex algorithms in $\mathcal{H}$.
  if $t_2 \le r - W + 1$ **then**
    Delete $\mathcal{A}_1^{(1)}$, $\mathcal{A}_1^{(2)}$, and t_i from $\mathcal{H}$.
    Reindex algorithms in $\mathcal{H}$.
Let s be the sampled output of $\mathcal{A}_1^{(2)}$ and let c be the number of times s has appeared afterwards.
if s has timestamp after $t - W + 1$ **then**
  Let F be the output of $\mathcal{A}_1^{(1)}$.
  return s with probability $\frac{c^p - (c-1)^p}{pF^{p-1}}$ ▷$p > 1$

Theorem 1.4. *For the insertion-only streaming model and $p \ge 1$, there exists a truly perfect L_p sampler that uses $\mathcal{O}(1)$ update time and $\mathcal{O}\left(n^{1-1/p} \operatorname{polylog}(n)\right)$ bits of space.*

Proof. Note that Sampler contains not only the updates in the sliding window, but an additional number of elements, since the substream actually starts before $t - W + 1$. Because $F_1 = W$ is the number of elements in the substream, then Sampler selects each item inside the window with probability $\frac{1}{F_1}$. Let $\mathcal{E}$ be the event that Sampler selects an element inside the window, which occurs with probability $\frac{W}{F_1} \ge \frac{1}{2}$, since $F_1 < 2W$. Conditioned on $\mathcal{E}$, then Sampler selects each item of the window with probability $\frac{1}{W}$. Hence if $i \in [n]$ appears f_i times inside the window, then Sampler outputs i with probability $\frac{f_i}{W}$, conditioned on $\mathcal{E}$.

The probability that the j^{th} particular instance of i inside the window is selected is $\frac{1}{W}$, conditioned on $\mathcal{E}$. Moreover for $p > 1$, the number of instances of i appearing after j, inclusive, is $f_i - j + 1$ so the probability that i is output is

$$\sum_{j=1}^{f_i} \frac{1}{W} \frac{(f_i - j + 1)^p - (f_i - j)^p}{pF^{p-1}} = \frac{f_i^p}{pWF^{p-1}},$$

where $F_p^{\frac{1}{p}} < F < 2F_p^{\frac{1}{p}}$ by Theorem A.4, with $F_p = \sum f_i^p$. Note that $(f_i - j + 1)^p - (f_i - j)^p \le pF^{p-1}$

for all $j \in [f_i]$, so returning s with probability $\frac{c^p-(c-1)^p}{pF^{p-1}}$ is a valid procedure.

Since $F_p^{\frac{1}{p}} \leq F_1 = W \leq W^{1-\frac{1}{p}} F_p^{\frac{1}{p}}$ for $p > 1$, then the probability that some sample is output is at least

$$\sum \frac{f_i^p}{pWF^{p-1}} \geq \frac{F_p}{p\left(W^{1-\frac{1}{p}}F_p^{\frac{1}{p}}\right)\left(2F_p^{\frac{1}{p}}\right)^{p-1}} \geq \frac{1}{p2^{p-1}W^{1-\frac{1}{p}}}.$$

Thus by repeating the sampler $\mathcal{O}\left(W^{1-\frac{1}{p}}\right)$ times, the algorithm will output a sample s with probability at least $\frac{2}{3}$. Moreover, the sample s will equal each index $i \in [n]$ with probability $\frac{f_i^p}{F_p}$. Since each sampler requires $\mathcal{O}(\log W + \log n)$ bits of space, then the space complexity follows, under the assumption that $\mathcal{O}(\log n) = \mathcal{O}(\log W)$. □

Jayaram and Woodruff [JW18b] give a perfect L_p sampler in the streaming model that uses $\mathcal{O}\left(\log^2 n \log \frac{1}{\delta}\right)$ space for $0 < p < 1$ to obtain $1-\delta$ probability of success. Their algorithm takes the underlying universe and duplicates each item $\text{poly}(n)$ times. As the stream arrives, an update to each element of the original stream is transformed into updates to each of the $\text{poly}(n)$ duplicated items. A linear transformation is then performed by scaling each duplicated item by the inverse of an exponential random variable. A sketch is maintained throughout the stream to output the item with the largest frequency at the end of the stream if it is sufficiently large. In particular, if z_0 is the frequency of the scaled duplicated item with the largest frequency and z' is the frequency vector of the scaled duplicated items *excluding* z_0, then the item will be reported if $|z_0| > 20\|z'\|_2$. Otherwise, the algorithm does not output anything, but multiple instances of the algorithm are run in parallel to ensure that some instance reports an item with probability at least $1-\delta$.

B Fast Perfect L_p Sampler for $0 < p < 1$ on Sliding Windows

In this section, we give a construction of a perfect L_p sampler (but not truly perfect) for $p \in (0,1)$ on sliding windows. As a specific case, it also provides a perfect L_p sampler in insertion-only streaming model that has faster update time than existing constructions, e.g. [JW18b].

Recall that an exponential random variable E is parametrized by a rate $\lambda > 0$ if it has cumulative distribution function $\mathbf{Pr}[E < x] = 1 - e^{-\lambda x}$.

Fact B.1 (Scaling of exponentials)**.** *Let E be an exponential random variable with rate $\lambda > 0$ and let $\alpha > 0$. Then αE is an exponential random variable with rate $\frac{\lambda}{\alpha}$.*

For a vector z, we define the anti-ranks to describe the indices of z whose corresponding frequencies are non-increasing in magnitude.

Definition B.2. *Let the anti-ranks $D(\cdot)$ be defined so that $|z_{D(1)}| \geq |z_{D(2)}| \geq \ldots \geq |z_{D(n^c)}|$. We use the notation $z_{-D(1)}$ to denote the vector z whose largest entry has been set to zero, i.e., $z_{D(1)} = 0$.*

Lemma B.3. *[Nag06] Let $z_1, \ldots, z_n$ be independent exponential random variables, so that z_i has rate $\lambda_i > 0$ for each $i \in [n]$. Then for any $i \in [n]$, we have*

$$\mathbf{Pr}[D(1) = i] = \frac{\lambda_i}{\sum_{j\in[n]} \lambda_j}.$$

Algorithm 7 Perfect L_p sampler for the sliding window model on insertion only streams and $p < 1$.

Input: A stream of updates $u_1, u_2, \ldots, u_t$, where each $u_i \in [n]$ represents a single update to a coordinate of the underlying vector f, and a size W for the sliding window.

Let α be a sufficiently large constant so that $n^\alpha \gg \text{poly}(n, W)$.
for $k = 1$ to $k = \alpha \log n$ **do**
    $\mathcal{S}_k \leftarrow \emptyset$
Let $e_{i,j}$ be an exponential random variable for each $i \in [n], j \in [n^c]$.
for each update u_r **do**
    for $i \in [n^c]$ **do**
        for $k = 1$ to $k = \alpha \log n$ **do**
            Generate $\frac{1}{e_{u_r,i}^{1/p}}$ instances of duplicated variable $z_{u_r,i}$.
            if $|\mathcal{S}_k| < 400\alpha c \log n$ **then**
                Add each instance into $\mathcal{S}_k$ with probability $\frac{100c \log n}{2^k}$ with timestamp r.
            Delete elements of $\mathcal{S}_k$ with timestamp less than $r - W$.
Let $\widehat{f}$ be a 2-approximation to the number of instances of duplicated variables in the active window. ▷Theorem A.5
Let k be the integer with $2^k \le \widehat{f} < 2^{k+1}$
if there exists (i, j) such that $z_{i,j}$ forms a majority of $\mathcal{S}_k$ **then**
    return i.
else
    Output FAIL.

We abuse notation and represent z in Algorithm 7 as both a frequency vector with n^{c+1} coordinates as well as a doubly indexed set $z_{i,j}$ with $i \in [n]$ and $j \in [n^c]$, to denote that each coordinate $i \in [n]$ is duplicated n^c times. To analyze Algorithm 7, we first recall the following lemma from [JW18b], which decomposes the value of the maximum coordinate of the duplicated vector into a large component that is independent of the index that achieves the max and a negligible component that depends on the index.

Lemma B.4. *[JW18b] For each $1 \le k < n^c - n^{9c/10}$, $p \in (0, 2]$ and $\nu \ge n^{-c/60}$, we have with probability $1 - \mathcal{O}\left(e^{-n^{c/3}}\right)$ that $|z_{D(k)}| = U_{D(k)}(1 + V_{D(k)})$, for $|V_{D_k}| = \mathcal{O}(\nu)$ and*

$$U_{D_k} = \left[\left(1 \pm \mathcal{O}\left(n^{-c/10}\right)\right) \sum_{\tau=1}^{k} \frac{E_\tau}{\mathbb{E}\left[\sum_{j=1}^{n^c} F_{D(j)}^p\right]}\right]^{-1/p},$$

where the E_τ are identically independently distributed exponential random variables that are independent of the value of $D(k)$ and F is the underlying frequency vector on the universe of n^c duplicated items.

The following lemma from [JW18b] states that with constant probability, the max coordinate of z in Algorithm 7 will dominate the p-norm of z.

Lemma B.5. *[JW18b] For $p < 2$, $z_{D(1)} > 20 \left\|z_{-D(1)}\right\|_p$ with constant probability.*

We now show that the probability that Algorithm 7 fails only negligibly depends on the index that achieves the max. This negligible difference can be absorbed into an $\frac{1}{\text{poly}(n)}$ additive component for the sampling probabilities.

Lemma B.6. *Let $\mathcal{E}$ denote the event that Algorithm 7 fails. Then* $\mathbf{Pr}\left[\mathcal{E} \mid z_{D(1)}\right] = \mathbf{Pr}\left[\mathcal{E}\right] + \frac{1}{\text{poly}(n)}$.

Proof. Consider Algorithm 7. We first analyze the probability that the algorithm fails, conditioned on a given value of $D(1)$. By Lemma B.4, we can rewrite $|z_{D(1)}| = U_{D(1)}(1 + V_{D(1)})$, where U_{D_1} is independent of the value of $D(1)$ and $|V_{D_1}| = \mathcal{O}(\nu)$. Thus the probability that the algorithm fails conditioned on the value of $D(1)$ is at most an additive $\mathcal{O}(\nu)$ amount from the probability that the algorithm fails. Hence, it suffices to set $\nu = \frac{1}{\text{poly}(n)}$.

To evaluate the probability that the algorithm fails, note that Lemma B.5 implies $|z_{D(1)}| > 20\left\|z_{-D(1)}\right\|_p > 20\left\|z_{-D(1)}\right\|_1$ with constant probability for $p < 1$. Let $2^k \leq \|z\|_1 \leq 2^{k+1}$ and let $\beta = \frac{\|z\|_1}{2^k} \geq 1$. If each element is sampled with probability $\frac{100c\log n}{2^k}$, then $90c\beta\log n$ instances of $D(1)$ will be sampled in expectation. By Chernoff bounds, at least $60c\beta\log n$ samples of $D(1)$ will be drawn with probability at least $1 - n^{-5c}$. On the other hand, at most $10c\beta\log n$ instances of other elements will be sampled in expectation. The probability less than $40c\beta\log n$ instances of other elements are sampled is at least $1 - n^{-5c}$. Thus $D(1)$ will be output with probability at least $1 - n^{-4c}$. □

We highlight that Lemma B.6 implicitly but crucially uses the fact that each sample is unbiased. That is, we cannot blindly interpret Lemma B.5 as claiming that it suffices to find the heavy-hitter that dominates p-norm of the duplicated frequency vector. Notably, naïvely using a sliding window heavy-hitter algorithm such as [BGL+18] to identify the index $i \in [n]$ that achieves the max does not work because these algorithms are biased against the items appearing near the boundary of the sliding window. For example, any heavy-hitter algorithm including a recently expired insertion to coordinate i is *more likely* to identify i as a heavy-hitter, so that the probability of failure is not independent of which coordinate achieves the max since it may be non-negligibly lower for i. Similarly, any heavy-hitter algorithm that excludes an active insertion to coordinate i is less likely to identify i as a heavy-hitter and again, the probability of failure is not independent of which coordinate achieves the max since it may be non-negligibly higher for i. Moreover, if a reduction from Lemma B.5 to the heavy-hitters problem in the sliding window were immediately true, then it would be possible to use a heavy-hitter sliding window algorithm such as [BGL+18] to do perfect L_p sampling in polylog(n) space.

Since Lemma B.6 states that the value of $z_{D(1)}$, i.e., the coordinate that achieves the max after the scaling of the exponential random variables, only affects the failure probability of the algorithm by a negligible amount, it remains to show that each coordinate i achieves the max with probability $\frac{|f_i|^p}{\sum_{j\in[n]}|f_j|^p}$.

Theorem B.7. *For $p < 1$, there exists a perfect L_p sampler for the sliding window model with insertion-only updates that uses $\mathcal{O}\left(\log^3 W\right)$ bits of space.*

Proof. Let $f \in \mathbb{R}^n$ be the underlying frequency vector implicitly defined by the sliding window. Each $i \in [n]$ is associated with n^c variables $z_{i,j}$, where $j \in [n^c]$. Since i is updated f_i times in the active window, then we have $z_{i,j} \sim \frac{f_i}{E_{i,j}^{1/p}}$, where $E_{i,j}$ is an exponential random variable. Thus by Fact B.1, each $|z_{i,j}|^{-p}$ is an exponential random variable with rate $|f_i|^p$. By Lemma B.3, the

probability that $z_{i,j}$ is the largest scaled variable across all $i \in [n]$, $j \in [n^c]$ is $\frac{|f_i|^p}{n^c \cdot \sum_{k\in[n]} |f_k|^p}$. Hence for a fixed $i \in [n]$, the probability that some variable $z_{i,j}$ achieves the max for some $j \in [n^c]$ is

$$\frac{n^c \cdot |f_i|^p}{n^c \cdot \sum_{k\in[n]} |f_k|^p} = \frac{|f_i|^p}{\sum_{k\in[n]} |f_k|^p}.$$

Now conditioned on some $z_{i,j}$ achieving the max, the probability that Algorithm 7 succeeds and correctly outputs i is only perturbed by an additive $\frac{1}{\text{poly}(n)}$ by Lemma B.6. Thus for each $i \in [n]$, the probability that Algorithm 7 outputs i is

$$\frac{|f_i|^p}{\sum_{k\in[n]} |f_k|^p} + \frac{1}{\text{poly}(n)}.$$

Hence, Algorithm 7 is a perfect L_p sampler.

To analyze the space complexity, observe that Algorithm 7 maintains $\mathcal{O}(\log n)$ samples in each of the sets $\mathcal{S}_1, \ldots, \mathcal{S}_{\alpha \log n}$. Each sample uses $\mathcal{O}(\log n)$ bits to store. Hence, these components of the algorithm use $\mathcal{O}(\log^3 n)$ bits of space in total and it remains to derandomize the exponential random variables, which we argue below. □

Derandomization of the Algorithm. [JW18b] used a combination of Nisan's pseudorandom generator (PRG) [Nis92] and a PRG of [GKM18] that fools a certain class of Fourier transforms to develop an efficient PRG that fools half-space testers. Using this half-space tester, [JW18b] showed that any streaming algorithm that only stores a random sketch $\mathbf{A} \cdot f$, with bounded independent and identically distributed entries that can be efficiently sampled, on an input stream whose intermediate underlying vectors have polynomially bounded entries can be efficiently derandomized. This suffices for the perfect L_p sampler in the streaming model, which uses a series of linear sketches to sample an output. Algorithm 7 uses a sampling based approach instead of linear sketches, so we must repurpose the derandomization of the perfect sampler from [JW18b]. Fortunately, we argue that a simple application of Nisan's PRG suffices to derandomize our algorithm.

Theorem B.8 (Nisan's PRG). *[Nis92] Let $\mathcal{A}$ be an algorithm that uses $S = \Omega(\log n)$ space and R random bits. Then there exists a pseudorandom generator for $\mathcal{A}$ that succeeds with high probability and runs in $\mathcal{O}(S \log R)$ bits.*

Recall that Nisan's PRG can be viewed as generating a stream of pseudorandom bits in a read-once tape that can be used to generate random variables to fool a small space tester. However, an input tape that can only be read once cannot be immediately given to algorithm to generate the exponential random variables $e_{i,j}$ and subsequently discarded because the value assigned to each $e_{i,j}$ must be consistent whenever the coordinate i is updated. Instead, we use the standard reordering trick to derandomize using Nisan's PRG.

For any fixed randomness $\mathbb{R}$ for the exponential random variables, let $\mathcal{T}_{\mathbb{R}}$ be the tester that tests whether our L_p sampler would output an index $i \in [n]$ if $\mathbb{R}$ is hard-coded into the tester and the random bits for the sampling procedures arrive in the stream. Specifically, we define $\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}, \mathcal{A}_1) = 1$ if the algorithm with access to independent exponential random variables outputs i on stream $\mathcal{S}$ and $\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}, \mathcal{A}_1) = 0$ otherwise. Similarly, we define $\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}, \mathcal{A}_2) = 1$ if using Nisan's PRG on our algorithm outputs i on stream $\mathcal{S}$ and $\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}, \mathcal{A}_2) = 0$ otherwise.

Let $\mathcal{S}_1$ be any fixed input stream and $\mathcal{S}_2$ be an input stream in which all updates to a single coordinate of the underlying frequency vector arrive consecutively. Observe that using Nisan's PRG on the algorithm suffices to fool $\mathcal{A}_{\mathbb{R}}$ on $\mathcal{S}_2$ from an algorithm with access to independent exponential random variables. That is, for all $i \in [n]$, we have

$$|\mathbf{Pr}\left[\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}_2, \mathcal{A}_1) = 1\right] - \mathbf{Pr}\left[\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}_2, \mathcal{A}_2) = 1\right]| = \frac{1}{\text{poly}(n)}.$$

On the other hand, the order of the inputs does not change the distribution of the outputs of the idealized process, so that

$$\mathbf{Pr}\left[\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}_1, \mathcal{A}_1) = 1\right] = \mathbf{Pr}\left[\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}_2, \mathcal{A}_1) = 1\right].$$

Similarly, the order of the inputs does not change the distribution of the outputs of the algorithm following Nisan's PRG, so that

$$\mathbf{Pr}\left[\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}_1, \mathcal{A}_2) = 1\right] = \mathbf{Pr}\left[\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}_2, \mathcal{A}_2) = 1\right].$$

Hence, we have

$$|\mathbf{Pr}\left[\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}_1, \mathcal{A}_1) = 1\right] - \mathbf{Pr}\left[\mathcal{T}_{\mathbb{R}}(i, \mathcal{S}_1, \mathcal{A}_2) = 1\right]| = \frac{1}{\text{poly}(n)}.$$

In other words, using Nisan's PRG on the algorithm on the $\mathcal{S}_1$ suffices to fool $\mathcal{A}_{\mathbb{R}}$ from an algorithm with access to independent exponential random variables.

Since our algorithm uses $\mathcal{O}\left(\log^3 n\right)$ bits of space and $\mathcal{O}\left(n\right)$ bits of randomness, then a naïve application of Nisan's PRG would derandomize our algorithm using $\mathcal{O}\left(\log^4 n\right)$ bits of space by Theorem B.8. Instead, we note that only the samples in set S_k with $2^k \leq \|z\|_1 \leq 2^{k+1}$ are tested by each our algorithm. Thus there exists a space $\mathcal{O}\left(\log^2 n\right)$ tester for our algorithm, which combined with Nisan's PRG, yields a $\mathcal{O}\left(\log^3 n\right)$ space derandomization of Algorithm 7, by Theorem B.8.

B.1 Insertion-Only Streams

Algorithm 8 Perfect L_p sampler for the streaming model on insertion only streams and $p < 1$.

Input: A stream of updates $u_1, u_2, \ldots, u_t$, where each $u_i \in [n]$ represents a single update to a coordinate of the underlying vector f, and a size W for the sliding window.

Let $e_{i,j}$ be an exponential random variable for each $i \in [n], j \in [n^c]$.
for each update u_r **do**
    for $i \in [n^c]$ **do**
        Insert $\frac{1}{e_{u_r,i}^{1/p}}$ instances of duplicated variable $z_{u_r,i}$ into stream S'.
        Run MisraGries on stream S'.
Let m be the stream length of S'
if MisraGries reports an item i with frequency at least $\frac{1}{2}m$. **then**
    return i.
else
    Output FAIL.

Theorem B.9. *For $p < 1$, there exists a perfect L_p sampler for the streaming model with insertion-only updates that uses $\mathcal{O}(\log n)$ bits of space.*

Proof. Consider Algorithm 8. We fix the value of $D(1)$ and analyze the probability that the algorithm fails. Lemma B.4 again implies that we can rewrite $|z_{D(1)}| = U_{D(1)}(1 + V_{D(1)})$, where U_{D_1} is independent of the value of $D(1)$ and $|V_{D_1}| = \mathcal{O}(\nu)$. Hence, the value of $D(1)$ only perturbs the probability that the algorithm fails by at most an additive $\mathcal{O}(\nu)$. Thus we set $\nu = \frac{1}{\text{poly}(n)}$ and absorb the additive error into the $\frac{1}{\text{poly}(n)}$ sampling error.

To evaluate the probability that the algorithm fails, note that Lemma B.5 implies $z_{D(1)} > 20\left\|z_{-D(1)}\right\|_p > 20\left\|z_{-D(1)}\right\|_1$ with constant probability for $p < 1$. Now conditioned on $z_{D(1)} > 20\left\|z_{-D(1)}\right\|_1$, a MisraGries data structure with $\epsilon = \frac{1}{100}$ in Theorem 3.2 will always include $D(1)$ in the list of indices. Moreover, MisraGries will output an estimated frequency for $z_{D(1)}$ larger than $\frac{1}{2}\|z\|_1$. Hence, $D(1)$ will always be output by the algorithm. Since MisraGries with $\epsilon = \mathcal{O}(1)$ uses $\mathcal{O}(\log n)$ bits of space, then the algorithm uses $\mathcal{O}(\log n)$ bits of space in total. □

B.2 Fast Update Time

Observe that we expect to sample $\mathcal{O}(\log n)$ elements at all times within the stream. Hence, it is wasteful to generate all variables $z_{u_r,j}$ for $j \in [n^c]$ each time a new update u_r arrives. Instead, we use the fact that the sum of exponential random variables converges to a p-stable distribution.

Theorem B.10. *[Hal81] Let $e_1, \ldots, e_{n^c}$ be exponential random variables with rate 1. Let $\beta = \frac{1}{2p} - \frac{1}{2}$ for $p < 1$ and $\beta = \frac{1}{p} - \frac{1}{2}$ for $1 < p < 2$. Then*

$$\mathbf{Pr}\left[\sum_{i=1}^{n^c} \frac{1}{e_i^{1/p}} \leq n^{c/p}x\right] = A_1(x) + E(x),$$

where $A_1(x)$ is the probability distribution functions of a p-stable random variable whose characteristic function can be explicitly computed and $E(x) = \mathcal{O}\left(\frac{1}{n^{c\beta}}\right)$.

Now instead of generating $\frac{1}{e_{i,j}^{1/p}}$ for each $i \in [n]$ and $j \in [n^c]$, we can instead generate a p-stable random variable C_i for each $i \in [n]$ according to Theorem B.10 that is approximately $\sum_{j=1}^{n^c} \frac{1}{e_{i,j}^{1/p}}$. We can then sample on the variables C_i rather than the individual $e_{i,j}$. When a variable C_i is sampled, we compute which variables $e_{i,j}$ the sampled variable C_i corresponds to at the end of the stream, using $\mathcal{O}(n^c)$ post-processing time.

Lemma B.5 observes that for $|z_{D(1)}| > 20\left\|z_{-D(1)}\right\|_p$ with constant probability for any $p < 2$. In particular for $p < 1$, then $|z_{D(1)}| > 20\left\|z_{-D(1)}\right\|_p > 20\left\|z_{-D(1)}\right\|_1$, so $z_{D(1)}$ is a heavy-hitter of the frequency vector F of all $z_{i,j}$. Thus $|C_{D(1)}| > 20\left\|C_{-D(1)}\right\|_1$ so that $C_{D(1)}$ is a heavy-hitter of the frequency vector C and so an instance of CountMin that finds the $\mathcal{O}(1)$ heavy-hitters of C with probability at least $1 - \frac{1}{n^c}$ will find $C_{D(1)}$ and identify it as the maximal element. The algorithm can then report $D(1)$.

Similarly, for $1 < p < 2$, we have $|z_{D(1)}| > 20\left\|z_{-D(1)}\right\|_2$, so $z_{D(1)}$ is an L_2 heavy-hitter of the frequency vector F of all $z_{i,j}$. Now we have $C_{D(1)} > 20\left\|C_{-D(1)}\right\|_2$ so that $C_{D(1)}$ is a heavy-hitter of the frequency vector C and so an instance of CountMin that finds the $\mathcal{O}(1)$ heavy-hitters of C with probability at least $1 - \frac{1}{n^c}$ will find $C_{D(1)}$ and identify it as the maximal element.

A similar argument can be used to derandomize the fast update time algorithm, by replacing the hard-coded randomness for the independent random variables in the tester with the hard-coded randomness for the independent p-stable random variables.

Corollary B.11. *For $p < 1$, there exists a perfect L_p sampler for the streaming model with insertion-only updates that uses $\mathcal{O}(\log n)$ bits of space,* $\text{polylog}(n)$ *update time, and* $\text{poly}(n)$ *post-processing time.*

C Truly Perfect Sampling on Random Order Streams

In this section, we first consider truly perfect L_2 sampling on random order streams using $\mathcal{O}\left(\log^2 n\right)$ bits of space and generalize this approach to truly perfect L_p samplers on random order streams for integers $p > 2$ using $\mathcal{O}\left(W^{1-\frac{1}{p-1}} \log n\right)$ bits of space. For $p = 2$, the idea is to consider two adjacent elements and see if they collide. Intuitively, an arbitrary position in the window is item i with probability $\frac{f_i}{W}$ due to the random order of the stream. Then the next position in the window is also item i with probability $\frac{f_i - 1}{W-1}$. The probability the two positions are both i is $\frac{f_i(f_i-1)}{W(W-1)}$, which is not quite the right probability.

Thus, we "correct" this probability by sampling item i in a position with probability $\frac{1}{W}$. Otherwise, with probability $1 - \frac{1}{W}$, we sample item i if the item is in the next position as well. Now the probability of sampling i on the two adjacent elements is $\frac{1}{W}\frac{f_i}{W} + \frac{W-1}{W}\frac{f_i}{W}\frac{f_i-1}{W-1} = \frac{f_i^2}{W^2}$. Hence if these two positions outputs some item, then the item is drawn from the correct distribution. Moreover, since we maintain the items and their positions, we can expire our samples whenever the items become expired. We can also output a random sample in the case there are multiple samples. We perform this procedure over all disjoint adjacent pairs in the stream and show that the algorithm will report some sample with constant probability, since we show that the expected number of samples and the squared expected number of samples is within a constant factor. We give the algorithm in Algorithm 9.

Remark C.1. *We remark that although our results are presented in the sliding window model, they can naturally apply to random-order insertion-only streams as well.*

Lemma C.2. *For each pair u_{2i-1} and u_{2i} with $2i - 1 > t - W$, the probability that Algorithm 9 samples $j \in [n]$ is $\frac{f_j^2}{W^2}$.*

Proof. The probability that u_{2i-1} is $\frac{f_j}{W}$ and u_{2i-1} is sampled with probability $\frac{1}{W}$. Otherwise, $u_{2i} = u_{2i-1} = j$ with probability $\frac{f_j(f_j-1)}{W(W-1)}$. Hence, the probability that Algorithm 9 samples j is

$$\frac{1}{W}\frac{f_j}{W} + \frac{W-1}{W}\frac{f_j(f_j-1)}{W(W-1)} = \frac{f_j^2}{W^2}.$$

□

We use the following formulation of the Paley-Zygmund Inequality.

Algorithm 9 Truly perfect L_2 sampler algorithm for the sliding window model on random order streams.

Input: A stream of updates $u_1, u_2, \ldots, u_t$, a size W for the sliding window. Let each $u_i \in [n]$ represent a single update to a coordinate of the underlying vector f.

$\mathcal{S} \leftarrow \emptyset$
for each update u_{2i-1} and u_{2i} **do**
  Draw $\phi \in [0,1]$ uniformly at random.
  if $\phi < \frac{1}{W}$ **then** ▷With probability $\frac{1}{W}$
    $\mathcal{S} \leftarrow \mathcal{S} \cup (u_{2i-1}, 2i-1)$.
  else ▷With probability $1 - \frac{1}{W}$
    if $u_{2i-1} = u_{2i}$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup (u_{2i-1}, 2i-1)$.
  if $(u_j, j) \in \mathcal{S}$ with $j \leq 2i - W$ **then** ▷(u_j, j) has expired
    Delete (u_j, j) from $\mathcal{S}$
  Let C be a sufficiently large constant so that $n^C > W$.
  if $|\mathcal{S}| > 2C \log n$ **then**
    Delete $C \log n$ elements uniformly at random from $\mathcal{S}$.
Return $(u_j, j) \in \mathcal{S}$ uniformly at random.

Theorem C.3 (Paley-Zygmund Inequality)**.** *Let $X \geq 0$ be a random variable with finite variance and $0 \leq \delta \leq 1$. Then*

$$\mathbf{Pr}\left[X > \delta \mathbb{E}\left[X\right]\right] \geq (1-\delta)^2 \frac{\mathbb{E}\left[X\right]^2}{\mathbb{E}\left[X^2\right]}.$$

Lemma C.4. *Algorithm 9 outputs some sample with probability at least $\frac{2}{3}$.*

Proof. By Lemma C.2, the probability that Algorithm 9 samples some $j \in [n]$ for a pair u_{2i-1} and u_{2i} with $2i - 1 > t - W$ is $\frac{F_2}{W^2}$, where $F_2 = \sum_{j=1}^{n} f_j^2$. We suppose t and $W \geq 4$ are even for the purpose of presentation, but a similar argument suffices for other parities of t and W. Let $X_1, \ldots, X_{\frac{W}{2}}$ be indicator variables so that $X_i = 1$ if the i^{th} disjoint consecutive pair in the window $u_{t-W+2i-1}, u_{t-W+2i}$ produced a sample and $X_i = 0$ otherwise and let $X = \sum X_i$. Thus the expected number of samples X inserted into $\mathcal{S}$ is

$$\mathbb{E}\left[X\right] = \sum \mathbb{E}\left[X_i\right] = \frac{W}{2} \frac{F_2}{W^2} = \frac{F_2}{2W}.$$

Moreover, we have $\mathbb{E}\left[X^2\right] = \sum \mathbb{E}\left[X_i X_j\right]$. For $i = j$, $\mathbb{E}\left[X_i X_j\right] = \frac{F_2}{W^2}$. For $i \neq j$, consider the probability the j^{th} disjoint pair is also sampled conditioned on $X_i = 1$. Formally, if $\tilde{f}_k$ is the frequency of k in the window excluding the i^{th} pair, then the probability that the j^{th} disjoint pair samples $k \in [n]$ is $\frac{\tilde{f}_k^2}{(W-2)^2}$ by a similar argument to Lemma C.2. Since $f_k \geq \tilde{f}_k$ and $\sum \tilde{f}_k = -2 + \sum f_k$, then $\mathbb{E}\left[X_j | X_i = 1\right]$ is within a constant factor of $\frac{F_2}{W^2}$. Hence, $\mathbb{E}\left[X^2\right]$ is within a constant factor of $\mathbb{E}\left[X\right]^2$ so then Algorithm 9 outputs some sample with constant probability by the Paley-Zygmund inequality. ∎

We now prove Theorem 1.6.

Theorem 1.6. *There exists a one-pass sliding window algorithm for random order insertion-only streams that outputs index $i \in [n]$ with probability $\frac{f_i^2}{F_2}$ and outputs **FAIL** with probability at most $\frac{1}{3}$, i.e., the algorithm is a truly perfect L_2 sampler, using $\mathcal{O}\left(\log^2 n\right)$ bits of space and $\mathcal{O}(1)$ update time.*

Proof. By Lemma C.4, Algorithm 9 outputs some sample with probability at least $\frac{2}{3}$. Conditioned on outputting a sample, Algorithm 9 outputs index $i \in [n]$ with probability $\frac{f_i^2}{F_2}$ by Lemma C.2. Since $\mathcal{S}$ maintains $\mathcal{O}(\log n)$ samples, each using $\log n + \log W$ bits, and $\log W = \mathcal{O}(\log n)$, then Algorithm 9 uses $\mathcal{O}\left(\log^2 n\right)$ bits of space. Note that for each update, at most one ordered pair is added to the set $\mathcal{S}$, which may be deleted at some later point in time. Thus the amortized update time is $\mathcal{O}(1)$. □

We now consider truly perfect L_p sampling for $p > 2$ on random order streams in the sliding window model. For truly perfect L_p sampling on random order streams for integers $p > 2$, the idea is to store consecutive blocks of $W^{1-\frac{1}{p-1}}$ elements in the stream, along with their corresponding timestamps. We then look for p-wise collisions within the block. The probability that a fixed group of p positions are all item i is $\frac{f_i}{W} \cdots \frac{f_i-p+1}{W-p+1}$.

We must therefore again "correct" the sampling probability so that the probability of sampling item i is proportional to f_i^p. To do this, we require the following fact so that we can write f_i^p as a positive linear combination of the quantities 1, f_i, $f_i(f_i-1)$, and so forth.

Lemma C.5. *For an integer $k \geq 0$, let $(x)_k$ denote the falling factorial so that $(x)_0 = 1$ and $(x)_k = x(x-1)\cdots(x-k+1)$. Let $S(n,k)$ denote the Stirling numbers of the second kind, e.g., the number of ways to partition a set of n objects into k non-empty subsets. Then for any integer $p \geq 0$, we have $x^p = \sum_{k=0}^{p} S(p,k)(x)_k$.*

Thus we can choose to accept the item i with some positive probability if the first k positions of the p positions are all item i. Otherwise, we can proceed to the first $k+1$ positions, accept the item i with some positive probability if they are all item i, and iterate. We give the algorithm in full in Algorithm 10.

Lemma C.6. *There exist $\alpha_1, \ldots, \alpha_p$ that are efficiently computable so that for each p-tuple in E, the probability that Algorithm 10 samples $j \in [n]$ is $\frac{f_j^p}{W^p}$.*

Proof. Let $v_1, \ldots, v_p$ be an ordered p-tuple in E. The probability that $v_1 = \ldots = v_p = j$ for some $j \in [n]$ is $\frac{f_j}{W} \cdots \frac{f_j-p+1}{W-p+1}$. Similarly the probability that $v_1 = \ldots = v_q = j$ for some $q \in [p]$ is $\frac{f_j}{W} \cdots \frac{f_j-q+1}{W-q+1}$. Thus by Lemma C.5, $\sum_{q=0}^{p} \alpha_q \frac{f_j}{W} \cdots \frac{f_j-q+1}{W-q+1} = \frac{f_j^p}{W^p}$ for $\alpha_q = S(p,q)\frac{W\cdots(W-q+1)}{W^p}$. □

Lemma C.7. *Algorithm 10 outputs some sample with probability at least $\frac{2}{3}$.*

Proof. By Lemma C.6, the probability that Algorithm 9 samples some $j \in [n]$ for a p-tuple in E is $\frac{F_p}{W^p}$, where $F_p = \sum_{j=1}^{n} f_j^p$. Let $X_1, \ldots, X_m$ be indicator variables so that $X_i = 1$ if the i^{th} p-tuple in the window in some fixed ordering produced a sample and $X_i = 0$ otherwise and let $X = \sum X_i$. Then $m = \left(\frac{W}{W^{1-\frac{1}{p-1}}}\right)\left(W^{1-\frac{1}{p-1}}\right)^p = W^{p-1}$. Thus the expected number of samples X inserted into $\mathcal{S}$ is $\Theta\left(\frac{F_p}{W}\right)$.

Algorithm 10 Truly perfect L_p sampler algorithm for the sliding window model on random order streams for $p > 2$.

Input: A stream of updates $u_1, u_2, \ldots, u_t$, a size W for the sliding window. Let each $u_i \in [n]$ represent a single update to a coordinate of the underlying vector f.

$\mathcal{S} \leftarrow \emptyset$
Store each disjoint block E of $W^{1-\frac{1}{p-1}}$ consecutive elements.
for each ordered p-tuple $(v_1, \ldots, v_p)$ of elements in E **do**
    for $i = 1$ to $i = p$ **do**
        if $v_1 = \ldots = v_i$ **then**
            Insert v_1 into $\mathcal{S}$ with probability $\frac{\alpha_i}{W^{p-1}}$.
if there exists a p-tuple in $\mathcal{S}$ with an expired element **then**
    Delete the p-tuple from $\mathcal{S}$
if $|\mathcal{S}| > 2W^{1-\frac{1}{p-1}}$ **then**
    Delete $W^{1-\frac{1}{p-1}}$ elements uniformly at random from $\mathcal{S}$.
Return a p-tuple from $\mathcal{S}$ uniformly at random.

Moreover, we have $\mathbb{E}\left[X^2\right] = \sum \mathbb{E}\left[X_{i_1} \ldots X_{i_p}\right]$, where $i_1, \ldots, i_p$ are contained in one of the disjoint consecutive block of $W^{1-\frac{1}{p-1}}$ elements. By a similar argument as Lemma C.4, $\mathbb{E}\left[X_{i_q} | X_{i_1}, \ldots, X_{i_{q-1}}\right]$ is within a constant multiple of $\mathbb{E}\left[X_{i_q}\right]$. Thus $\mathbb{E}\left[X^2\right]$ is within a constant factor of $\mathbb{E}\left[X\right]^2$ so then Algorithm 10 outputs some sample with constant probability by the Paley-Zygmund inequality. □

We now show the correctness of our general sliding window algorithm.

Theorem C.8. *Let $p > 2$ be a fixed integer. There exists a one-pass sliding window algorithm for random order insertion only streams that outputs index $i \in [n]$ with probability $\frac{f_i^p}{\sum_{j=1}^n f_j^p}$ with probability at least $\frac{2}{3}$, i.e., the algorithm is a truly perfect L_p sampler, using $\mathcal{O}\left(W^{1-\frac{2}{p}} \log n\right)$ bits of space.*

Proof. By Lemma C.7, Algorithm 10 outputs some sample with probability at least $\frac{2}{3}$. Conditioned on outputting a sample, Algorithm 10 outputs index $i \in [n]$ with probability $\frac{f_i^p}{\sum_{j=1}^n f_j^p}$ by Lemma C.6. Since $\mathcal{S}$ maintains $\mathcal{O}\left(W^{1-\frac{1}{p-1}}\right)$ samples, each using $\mathcal{O}\left(\log n\right)$ bits for $\log W = \mathcal{O}\left(\log n\right)$, then Algorithm 10 uses $\mathcal{O}\left(W^{1-\frac{1}{p-1}} \log n\right)$ bits of space. □

We remark that the time complexity of Algorithm 10 can be significantly improved for insertion-only data streams. Rather than enumerating over all possible p-tuples in a block of $\mathcal{O}\left(W^{1-\frac{1}{p-1}}\right)$ samples, it suffices to maintain the frequency g_i of each distinct coordinate $i \in [n]$ in the block, in order to simulate the sampling process of the p-wise collisions. Namely, we observe that although the final step of Algorithm 10 can be viewed as selecting a uniformly random element across all tuples ever inserted into $\mathcal{S}$, the same probability distribution holds for any sample uniformly inserted into $\mathcal{S}$ by a single block. Thus, we can equivalently instead sample a uniformly random element across all tuples inserted into $\mathcal{S}$ by each block, if the block inserts anything into $\mathcal{S}$ in the original process. That is, we can compute the size $k \in [p]$ of the k-tuple that represents the uniformly random element

inserted into $\mathcal{S}$ by each block, which only requires knowledge of the frequencies g_i for the block. Updating the frequency of each coordinates and subsequently updating the marginal probabilities for each index uses $\mathcal{O}(1)$ time per update. Thus, we obtain the following for insertion-only streams:

Theorem 1.7. *Let $p > 2$ be a fixed integer. There exists a one-pass algorithm that outputs an index $i \in [n]$ with probability $\frac{f_i^p}{\sum_{j=1}^n f_j^p}$, and outputs **FAIL** with probability at most $\frac{1}{3}$ on a random-order insertion-only stream of length m , i.e., the algorithm is a truly perfect L_p sampler, using $\mathcal{O}\left(m^{1-\frac{1}{p-1}} \log n\right)$ bits of space and $\mathcal{O}(1)$ update time.*

D Strict Turnstile Algorithms

In this section, we show that our techniques can be also be extended to obtaining truly perfect L_p samplers in the strict turnstile model, where updates to each coordinate may be both negative and positive integers but the underlying frequency vector at each point in time consists of non-negative coordinates. We assume the magnitude of each update in the stream to be at most $M = \text{poly}(n)$.

Theorem D.1 (Theorem 4 in [Gan08])**.** *There exists a deterministic streaming algorithm that outputs whether the underlying frequency vector in $\{0, \dots, M\}^n$ has more than k nonzero coordinates or fewer than $4k$ nonzero coordinates. The algorithm uses $\mathcal{O}(k \log(n/k) \log(Mn))$ bits of space and $\mathcal{O}\left(\log^2(n/k)\right)$ amortized time per arriving update.*

We remark that Theorem D.1 uses $\mathcal{O}\left(k \log^2(n/k)\right)$ time per arriving update as stated in [Gan08] due to performing a fast Vandermonde matrix-vector multiplication at each step. However, if we instead batch updates by storing the most recent k updates and then amortize performing the fast Vandermonde matrix-vector multiplication over the next k steps, then the amortized running time is $\mathcal{O}\left(\log^2(n/k)\right)$ as stated in Theorem D.1.

Theorem D.2 ([GM08, Gan08])**.** *There exists a deterministic streaming algorithm uses space $\mathcal{O}(k \log(Mn) \log(n/k))$ and update time $\text{polylog}(n/k)$ and recovers an underlying k-sparse frequency vector in $\{-M, \dots, M\}^n$.*

We similarly remark that Theorem D.2 as stated in [GM08, Gan08] uses $\mathcal{O}(k \log(n/k))$ time per arriving update due to maintaining counters tracking the degrees of $\mathcal{O}(k \log(n/k))$ vertices on the right side of a regular bipartite approximate lossless expander graph, where each vertex on the left corresponds to a coordinate of the universe and has degree $\text{polylog}(n/k)$. Moreover, the lossless expander graph is explicit, so computation of the neighborhood of each vertex uses $\text{polylog}(n/k)$ time [GUV09]. Thus if we again batch updates by storing the most recent k updates and then amortize updating the degrees of the $\mathcal{O}(k \log(n/k))$ vertices on the right side of the graph, then the amortized running time is $\text{polylog}(n/k)$ as stated in Theorem D.2.

By setting $k = 2\sqrt{n}$ in Theorem D.2, we can then form a set T of up to $8\sqrt{n}$ unique nonzero coordinates in the frequency vector, analogous to Algorithm 5. Similarly, setting $k = 2\sqrt{n}$ in Theorem D.1 tests whether the sparsity of the vector is at most $8\sqrt{n}$, in which case a random coordinate of T can be returned, or the sparsity of the vector is at least $2\sqrt{n}$, in which case an element of a random subset S of size $2\sqrt{n}$ as defined in Algorithm 5 will be a nonzero coordinate of the vector with constant probability. Thus we obtain the same guarantees as Theorem 5.2 for a strict turnstile stream.

Theorem D.3. *Given $\delta \in (0,1)$, there exists a truly perfect F_0 sampler on strict turnstile streams that uses $\mathcal{O}\left(\sqrt{n}\log^2 n \log\frac{1}{\delta}\right)$ bits of space and* $\operatorname{polylog} n \log\frac{1}{\delta}$ *update time and succeeds with probability at least $1-\delta$.*

We now show that we can get truly perfect L_p samplers in strict turnstile streams using $\mathcal{O}\left(n^{1-1/p+\gamma}\log n\right)$ space over $\mathcal{O}(1/\gamma)$ constant number of passes, where $\gamma > 0$ is a trade-off parameter. This shows a separation for L_p samplers between general turnstile streams and strict turnstile streams.

We first consider a truly perfect L_1 sampler. The idea is just to partition the universe $[n]$ into n^γ chunks so that for each $i \in [\gamma]$, the i-th chunk corresponds to the coordinates between $(i-1)n^\gamma + 1$ and in^γ. Our algorithm stores the sum of counts on each chunk in the first pass, which requires $\mathcal{O}(n^\gamma \log n)$ bits of space. We then sample a chunk proportional to this sum, so that the total universe size has decreased to be size $n^{1-\gamma}$. In the subsequent passes, we recursively partition the decreased universe into n^γ chunks as before and repeat. Thus after repeating $\mathcal{O}(1/\gamma)$, we sample a single coordinate under the desired distribution.

For general p, note that our algorithm for insertion-only is actually just sampling according to frequency. Thus we can use $\mathcal{O}(1/\gamma)$ passes as in the previous paragraph to do this, and then we can simulate our old algorithm with these samples. As the insertion-only algorithm uses $\mathcal{O}\left(n^{1-1/p}\right)$ samples, we require $\mathcal{O}\left(n^{1-1/p+\gamma}\log n\right)$ bits of space in $\mathcal{O}(1/\gamma)$ passes. Upon obtaining the samples, we also need to deterministically compute a number Z such that $\|f\|_\infty \le Z \le \|f\|_\infty + \frac{m}{n^{1-1/p}}$ for the frequency vector f. To do this, we can partition the universe $[n]$ into $n^{1-1/p+\gamma}$ chunks of size $n^{1/p-\gamma}$. There can be at most $n^{1-1/p}$ chunks whose coordinate sums are at least $\frac{m}{n^{1-1/p}}$. Thus, we have reduced the number of possible universe items by a factor of $1/n^\gamma$. Hence after $\mathcal{O}(1/\gamma)$ passes repetitions, we find any such item $\|f\|_\infty$ and thus obtain a deterministic estimate Z such that $\|f\|_\infty \le Z \le \|f\|_\infty + \frac{m}{n^{1-1/p}}$.

More generally, we have the following reduction:

Theorem 1.5. *Suppose there exists a truly perfect L_p sampler in the one-pass insertion-only streaming model that uses S bits of space. Then there exists a truly perfect L_p sampler that uses $\tilde{\mathcal{O}}(Sn^\gamma)$ space and $\mathcal{O}\left(\frac{1}{\gamma}\right)$ passes over a strict turnstile stream, which induces intermediate frequency vectors with nonnegative coordinates.*