SIAM J. COMPUT.
Vol. 50, No. 2, pp. 382–439

PERFECT L_p SAMPLING IN A DATA STREAM*

RAJESH JAYARAM† AND DAVID WOODRUFF†

Abstract. In this paper, we resolve the one-pass space complexity of *perfect* L_p sampling for $p \in (0,2)$ in a stream. Given a stream of updates (insertions and deletions) to the coordinates of an underlying vector $f \in \mathbb{R}^n$, a perfect L_p sampler must output an index i with probability $|f_i|^p/\|f\|_p^p$ and is allowed to fail with some probability δ. So far, for $p > 0$ no algorithm has been shown to solve the problem exactly using poly$(\log n)$-bits of space. In 2010, Monemizadeh and Woodruff introduced an *approximate* L_p *sampler* which, given an approximation parameter ν, outputs i with probability $(1\pm\nu)|f_i|^p/\|f\|_p^p$, using space polynomial in ν^{-1} and $\log(n)$. The space complexity was later reduced by Jowhari, Sağlam, and Tardos to roughly $O(\nu^{-p}\log^2 n \log\delta^{-1})$ for $p \in (0,2)$, which matches the general $p \geq 0$ lower bound of $\Omega(\log^2 n \log\delta^{-1})$ in terms of n and δ, but is loose in terms of ν. Given these nearly tight bounds, it is perhaps surprising that no lower bound exists in terms of ν—not even a bound of $\Omega(\nu^{-1})$ is known. In this paper, we explain this phenomenon by demonstrating the existence of an $O(\log^2 n \log\delta^{-1})$-bit *perfect* L_p sampler for $p \in (0,2)$. This shows that ν need not factor into the space of an L_p sampler, which closes the complexity of the problem for this range of p. For $p = 2$, our bound is $O(\log^3 n \log\delta^{-1})$-bits, which matches the prior best known upper bound of $O(\nu^{-2}\log^3 n \log\delta^{-1})$, but has no dependence on ν. Note that there is still a $\log n$ gap between our upper bound and the lower bound for $p = 2$, the ution of which we leave as an open problem. For $p < 2$, our bound holds in the random oracle model, matching the lower bounds in that model. However, we show that our algorithm can be derandomized with only a $O((\log\log n)^2)$ blow-up in the space (and no blow-up for $p = 2$). Our derandomization technique is quite general, and can be used to derandomize a large class of linear sketches, including the more accurate count-sketch variant of Minton and Price [*Proceedings of the* 25*th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, 2014, pp. 669–686], resolving an open question in that paper. Finally, we show that a $(1 \pm \epsilon)$ relative error estimate of the frequency f_i of the sampled index i can be obtained using an additional $O(\epsilon^{-p}\log n)$-bits of space for $p < 2$, and $O(\epsilon^{-2}\log^2 n)$ bits for $p = 2$, which was possible before only by running the prior algorithms with $\nu = \epsilon$.

Key words. streaming, sampling, algorithms

AMS subject classifications. 68W01, 68W20

DOI. 10.1137/18M1229912

1. Introduction.

The streaming model of computation has become increasingly important for the analysis of massive datasets, where the sheer size of the input imposes stringent restrictions on the resources available to algorithms. Examples of such datasets include internet traffic logs, sensor networks, financial transaction data, database logs, and scientific data streams (such as huge experiments in particle physics, genomics, and astronomy). Given their prevalence, there is a large body of literature devoted to designing extremely efficient one-pass algorithms for analyzing data streams. We refer the reader to [BBD+02, M+05] for surveys of these algorithms and their applications.

More recently, the technique of sampling has proven to be tremendously powerful for the analysis of data streams. Substantial literature has been devoted to the study of sampling for problems in big data [M+05, Haa16, Coh15, CDK+09, CDK+14,

*Received by the editors December 3, 2018; accepted for publication (in revised form) October 14, 2020; published electronically March 30, 2021. A preliminary version of this work appeared in the *Proceedings of the IEEE* 59*th Annual Symposium on Foundations of Computer Science (FOCS)*, 2018.

https://doi.org/10.1137/18M1229912

Funding: The authors are thankful for the partial support by the National Science Foundation under grant CCF-1815840.

†Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15217 USA (rkjayara@cs.cmu.edu, dwoodruf@cs.cmu.edu).

CCD11, EV03, GM98a, Knu98, MM12, Vit85a, CCD12, GLH08, GLH06], with applications to network traffic analysis [TLJ10, HNG+07, GKMS01, MCS+06, Duf04], databases [Olk93, Haa16, HNSS96, HS92, LNS90, LN95], distributed computation [WZ16, CMYZ10, CMYZ12, TW11], and low-rank approximation [WZ16, FKV04, DV06]. While several models for sampling in data streams have been proposed [BDM02, AKO10, CMYZ10], one of the most widely studied is the L_p samplers introduced in [MW10]. Roughly speaking, given a vector $f \in \mathbb{R}^n$, the goal of an L_p sampler is to return an index $i \in \{1, 2, \ldots, n\}$ with probability $|f_i|^p/\|f\|_p^p$. In the data stream setting, the vector f, called the *frequency vector*, is given by a sequence of 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 m updates of the form $(i_t, \Delta_t) \in [n] \times \{-M, \ldots, M\}$ for some integer $M > 0$ and $t \in [m]$. The update (i_t, Δ_t) causes the change $f_{i_t} \leftarrow f_{i_t} + \Delta_t$. This is known as the *general turnstile* model of streaming. A 1-pass L_p sampler must return an index given only one pass through the updates of the stream.

Since their introduction, L_p samplers have been utilized to develop alternative algorithms for important streaming problems, such as the heavy hitters problem, L_p estimation, cascaded norm estimation, and finding duplicates in data streams [AKO10, MW10, JST11, BOZ12]. For the case of $p = 1$ and insertion only streams, where the updates to f are strictly positive, the problem is easily solved using $O(\log n)$ bits of space with the well-known reservoir sampling algorithm [Vit85a]. When deletions (negative updates) to the stream are allowed or when $p \neq 1$, however, the problem is more complicated. In fact, the question of whether such samplers even exist was posed by Cormode, Murthukrishnan, and Rozenbaum in [CMR05]. Later on, Monemizadeh and Woodruff demonstrated that if one permits the sampler to be *approximately* correct, such samplers are indeed possible [MW10]. We formally state the guarantee given by an approximate L_p sampler below.

DEFINITION 1. *Let $f \in \mathbb{R}^n$ and $\nu \in [0, 1)$. For $p > 0$, an approximate L_p sampler with ν-relative error is an algorithm which returns an index $i \in \{1, 2, \ldots, n\}$ such that for every $j \in \{1, 2, \ldots, n\}$*

$$\mathbf{Pr}[i = j] = \frac{|f_j|^p}{\|f\|_p^p}(1 \pm \nu) + O(n^{-c}),$$

where $c \geq 1$ is some arbitrarily large constant. For $p = 0$, the problem is to return j with probability $(1 \pm \nu)\frac{\mathbf{1}_{f_j \neq 0}}{|\{j : f_j \neq 0\}|} + O(n^{-c})$, where $\mathbf{1}_{f_j \neq 0}$ is the indicator function, i.e., $\mathbf{1}_{f_j \neq 0} = 1$ if $f_j \neq 0$ and $\mathbf{1}_{f_j \neq 0} = 0$ otherwise. If $\nu = 0$, then the sampler is said to be perfect. An L_p sampler is allowed to output `FAIL` with some probability δ. However, in this case it must not output any index.

The one-pass approximate L_p sampler introduced in [MW10] requires $\text{poly}(\nu^{-1}, \log n)$ space, albeit with rather large exponents. Later on, in [AKO10], the complexity was reduced significantly to $O(\nu^{-p}\log^3(n)\log(1/\delta))$-bits[1] for $p \in [1, 2]$, using a technique known as *precision sampling*. Roughly, the technique of precision sampling consists of scaling the coordinates f_i by random variable coefficients $1/t_i$ as the updates arrive, resulting in a new stream vector $z \in \mathbb{R}^n$ with $z_i = f_i/t_i$. The

[1]We note that previous works [JST11, KNP+17] have cited the sampler of [AKO10] as using $O(\log^3(n))$-bits of space, but the space bound given in their paper is in *machine words* and is therefore a $O(\log^4(n))$ bit bound with $\delta = 1/\text{poly}(n)$. In order to obtain an $O(\log^3(n)\log(1/\delta))$ bit sampler, their algorithm must be modified to use fewer repetitions.

algorithm then searches for all z_i which cross a certain threshold T. Observe that if $t_i = u_i^{1/p}$, where u_i is uniform on $[0,1]$, then the probability that $f_i/t_i \geq T$ is precisely $\mathbf{Pr}[u_i < |f_i|^p/T^p] = |f_i|^p/T^p$. By running an L_p estimation algorithm to obtain $T \in [\frac{1}{2}\|f\|_p, \frac{3}{2}\|f\|_p]$, an L_p sampler can then return any i with $z_i \geq T$ as its output. These heavy coordinates can be found using any of the well-known η-heavy hitters algorithms for a sufficiently small precision η.

Using a tighter analysis of this technique with the same scaling variables $t_i = u_i^{1/p}$, Jowhari, Sağlam, and Tardos reduced the space complexity of L_p sampling for $p < 2$ to $O(\nu^{-\max\{1,p\}} \log^2(n) \log(1/\delta))$-bits for $p \in (0,2) \setminus \{1\}$, and $O(\nu^{-1} \log(\nu^{-1}) \log^2(n) \log(1/\delta))$ bits of space for $p = 1$ [JST11]. Roughly speaking, their improvements result from a more careful consideration of the precision η needed to determine when a z_i crosses the threshold, which they do via the tighter tail-error guarantee of the well-known count-sketch heavy hitters algorithm [CCFC02a]. In addition, they give an $O(\log^2(n) \log(1/\delta))$ perfect L_0 sampler and demonstrate an $\Omega(\log^2(n))$-bit lower bound for L_p samplers for any $p \geq 0$. Recently, this lower bound was extended to $\Omega(\log^2(n) \log(1/\delta))$ [KNP+17] bits, which closes the complexity of the problem for $p = 0$.

For $p \in (0,2)$, this means that the upper and lower bounds for L_p samplers are tight in terms of n, δ, but a gap exists in the dependency on ν. This being the case, it would seem natural to search for an $\Omega(\nu^{-p} \log^2(n) \log(1/\delta))$ lower bound to close the complexity of the problem. It is perhaps surprising, therefore, that no lower bound in terms of ν exists—not even an $\Omega(\nu^{-1})$ bound is known. This poses the question of whether the $\Omega(\log^2(n) \log(1/\delta))$ lower bound is in fact correct.

1.1. Our contributions. In this paper, we explain the phenomenon of the lack of an $\Omega(\nu^{-1})$ lower bound by showing that ν need not enter the space complexity of an L_p sampler at all. In other words, we demonstrate the existence of *perfect L_p samplers* using $O(\log^2(n) \log(1/\delta)(\log\log n)^2)$-bits of space for $p \in (0,2)$, thus resolving the space complexity of the problem up to $\log\log(n)$ terms.[2] In the random oracle model, where we are given random access to an arbitrarily long tape of random bits which do not count against the space of the algorithm, our upper bound is $O(\log^2(n) \log(1/\delta))$, which matches the lower bound in the random oracle model. For $p = 2$, our space is $O(\log^3(n) \log(1/\delta))$-bits, which matches the best known upper bounds in terms of n, δ, yet again has no dependency on ν. In addition, for $p < 2$ and the high probabiltiy regime of $\delta < 1/n$, we obtain a $O(\log^3(n))$-bit perfect L_p sampler, which also tightly matches the lower bound without paying the extra $(\log\log n)^2$ factor. A summary of the prior upper bounds for L_p sampling, along with the contributions of this work, is given in Figure 1.

In addition to outputting a perfect sample i from the stream, for $p \in (0,2)$ we also show that, conditioned on an index being outputted, given an additional additive $O(\min\{\epsilon^{-2}, \epsilon^{-p} \log(\frac{1}{\delta_2})\} \log(n) \log(1/\delta_2))$-bits we can provide a $(1 \pm \epsilon)$ approximation of the frequency $|f_i|$ with probability $1 - \delta_2$. This separates the space dependence on $\log^2(n)$ and ϵ for frequency approximation, allowing us to obtain a $(1 \pm \epsilon)$ approximation of $|f_i|$ in $O(\log^2(n) + \epsilon^{-p} \log(n))$ bits of space with constant

[2] A previous version of this work claimed $O(\log^2(n) \log(1/\delta))$ bits of space for $p < 2$, but contained an error in the derandomization. Thus, this bound only held in the random oracle model. In the present version we correct this derandomization using a slightly different algorithm, albeit with a $(\log\log n)^2$ blow-up in the space. The algorithm from the previous version can be found in Appendix A, along with a new analysis of its derandomization that allows it to run in $O(\log^2(n)(\log\log(n))^2)$-bits of space.

L_p sampling upper bound (bits)	p range	Notes	Citation
$O(\log^3(n))$	$p = 0$	perfect L_0 sampler, $\delta = 1/\text{poly}(n)$	[FIS08]
$O(\log^2(n)\log(1/\delta_2))$	$p = 0$	perfect L_0 sampler	[JST11]
$\text{poly}\log(\nu^{-1}, n)$	$p \in [0, 2]$	$\delta = 1/\text{poly}(n)$	[MW10]
$O(\nu^{-p}\log^3(n)\log(1/\delta))$	$p \in [1, 2]$	$(1 \pm \nu)$-relative error	[AKO10]
$O(\nu^{-\max\{1,p\}}\log^2(n)\log(1/\delta))$	$p \in (0, 2) \setminus \{1\}$	$(1 \pm \nu)$-relative error	[JST11]
$O(\nu^{-1}\log(\nu^{-1})\log^2(n)\log(1/\delta))$	$p = 1$	$(1 \pm \nu)$-relative error	[JST11]
$O(\log^2(n)\log(1/\delta))$	$p \in (0, 2)$	perfect L_p sampler, random oracle model, matches lower bound	this work
$O(\log^2(n)\log(1/\delta)(\log\log n)^2)$	$p \in (0, 2)$	perfect L_p sampler	this work
$O(\log^3(n)\log(1/\delta))$	$p = 2$	perfect L_2 sampler	this work
$O(\log^3(n))$	$p \in (0, 2)$	$\delta = 1/\text{poly}(n)$	this work

FIG. 1. *Evolution of one pass L_p sampling upper bounds, with the best known lower bound of $\Omega(\log^2(n)\log(1/\delta))$ for $p \geq 0$* [KNP+17] *(see also* [JST11] *for a lower bound for constant δ).*

probability, whereas before this required $O(\epsilon^{-p}\log^2(n))$ bits of space. For $p = 2$, our bound is $O(\epsilon^{-2}\log^2(n)\log(1/\delta_2))$, which still improves upon the prior best known bounds for estimating the frequency by an $O(\log(n))$-factor. Finally, we show an $\Omega(\epsilon^{-p}\log(n)\log(1/\delta_2))$ bits of space lower bound for producing the $(1 \pm \epsilon)$ estimate (conditioned on an index being returned).

General derandomization. Along the way to derandomizing our main L_p sampling algorithm, we develop a *generic* technique that allows for the black-box derandomization of a large class of linear sketches. This theorem will not be sufficient alone to derandomize our algorithm, but is suitable for a number of other applications. For instance, it provides the first efficient derandomization of the count-sketch variant of Minton and Price [MP14], a discussion of which can be found in section 5.2.3. We state the generic theorem here. In what follows, for a matrix A, let $\texttt{vec}(A)$ denote the vectorization of A, obtained by stacking the columns of A together.

Theorem 5. *Fix $n, t, k \geq 1$, and fix $f \in \{-M, \ldots, M\}^n$, where $M = \text{poly}(n)$. Let $X \in \mathbb{R}^{t \times nk}$ be any fixed matrix with entries contained within $\{-M, \ldots, M\}$, and let $\mathcal{D}$ be any distribution over matrices $A \in \mathbb{R}^{k \times n}$ such that the entries $A \sim \mathcal{D}$ are independent and identically distributed (i.i.d.) and can be sampled using $O(\log n)$-bits. Let $\sigma : \mathbb{R}^k \times \mathbb{R}^t \to \{0, 1\}$ be any function, and fix any constant $c \geq 1$. Then there is a distribution $\mathcal{D}'$ over matrices $A' \in \mathbb{R}^{k \times n}$ such that $A' \sim \mathcal{D}'$ can be generated via a random seed of length $O((k + t)\log(n)(\log\log n)^2)$, such that*

$$\left|\mathbf{Pr}\Big[\sigma(Af, X \cdot \texttt{vec}(A)) = 1\Big] - \mathbf{Pr}\Big[\sigma(A'f, X \cdot \texttt{vec}(A')) = 1\Big]\right| < n^{-c(k+t)}$$

and such that each entry of A' can be computed in time $\tilde{O}(1)$ using working space linear in the seed length.

In the above theorem, σ can be defined as a tester with $\sigma(Af, X \cdot \texttt{vec}(A)) = 1$ whenever a streaming algorithm depending only on Af and $X \cdot \texttt{Vec}(A)$ succeeds. Note that the matrix X allows an algorithm to also depend lightly on A, in addition to the linear sketch Af. For instance, $X \cdot \texttt{vec}(A)$ could store an entire column of A, as is needed for count-sketch. We believe Theorem 5 is an important step toward a universal derandomization of streaming algorithms.

We remark that many streaming algorithms, such as the p-stable sketches of Indyk [Ind06] for L_p estimation, depend only on a linear sketch Af. Specifically, for $0 < p \leq 2$, to estimate $\|f\|_p^p = \sum_{i \in [n]} |f_i|^p$ to relative error $(1 \pm \epsilon)$, the algorithm of [Ind06] generates a matrix A with $O(\epsilon^{-2})$ rows and entries drawn independently

from a p-stable distribution. The original derandomization of [Ind06] applied Nisan's pseudorandom generator (PRG) [Nis92], resulting in a $\log n$ blow-up in the space; namely, the algorithm required $O(\epsilon^{-2}\log^2 n)$ bits of space. This blow-up was shown to be unnecessary by Kane, Nelson, and Woodruff [KNW10], who gave an involved argument demonstrating that it suffices to generate the entries of A with limited independence, requiring a total of $O(\epsilon^{-2}\log n)$ bits. Theorem 5, therefore, gives an alternative derandomization of Indyk's p-stable sketches, although it requires a slightly suboptimal $O(\epsilon^{-2}\log n(\log\log n)^2)$ bits of space.

1.2. Applications. Since their introduction, it has been observed that L_p samplers can be used as a building block in algorithms for many important streaming problems, such as finding heavy hitters, L_p-norm estimation, cascaded norm estimation, and finding duplicates in data streams [AKO10, MW10, JST11, BOZ12]. L_p samplers, particularly for $p = 1$, are often used as a black-box subroutine to design representative histograms of f on which more complicated algorithms are run [GMP, GM98a, Olk93, GKMS02, HNG+07, CMR05]. For these black-box applications, the only property of the samplers needed is the distribution of their samples. Samplers with relative error are statistically biased and, in the analysis of more complicated algorithms built upon such samplers, this bias and its propagation over multiple samples must be accounted for and bounded. The analysis and development of such algorithms would be simplified dramatically, therefore, with the assumptions that the samples were truly uniform (i.e., from a perfect L_1 sampler). In this case, no error terms or variational distance need be accounted for. Our results show that such an assumption is possible without affecting the space complexity of the sampler.

Note that in Definition 1, we allow a perfect sampler to have n^{-c+1} variation distance to the true L_p distribution. We note that this definition is in line with prior work, observing that even the perfect L_0 sampler of [JST11] incurs such an error from derandomizing with Nisan's PRG. Nevertheless, this error will never be detected if the sampler is run polynomially many times in the course of constructing a histogram, and such a sampler is therefore statistically indistinguishable from a truly uniform sampler and can be used as a black-box.

Another motivation for utilizing perfect L_p samplers comes from applications in privacy. Here $f \in \mathbb{R}^n$ is some underlying dataset, and we would like to reveal a sample $i \in [n]$ drawn from the L_p distribution over f to some external party without revealing too much global information about f itself. Using an approximate L_p sampler introduces a $(1 \pm \nu)$ multiplicative bias into the sampling probabilities, and this bias can depend on global properties of the data. For instance, such a sampler might bias the sampling probabilities of a large set S of coordinates by a $(1 + \nu)$ factor if a certain global property P holds for f and may instead bias them by $(1 - \nu)$ if a disjoint property P' holds. Using only a small number of samples, an adversary would then be able to distinguish whether P or P' holds by determining how these coordinates were biased. On the other hand, the bias in the samples produced by a perfect L_p sampler is polynomially small, and thus the leakage of global information could be substantially smaller when using one.

One formalization of what it means to "leak" global information comes from the literature on *private approximations* [FIM+01, Woo11], where given a nonnegative real-valued function $g(x)$ for some input x, the goal is to output a $(1 \pm \nu)$ approximation R to $g(x)$, such that the value R reveals no more information about the input x than the actual value of $g(x)$ reveals about x. Specifically, to do this, one must show that the distribution of the value R can be simulated given only knowledge of the value

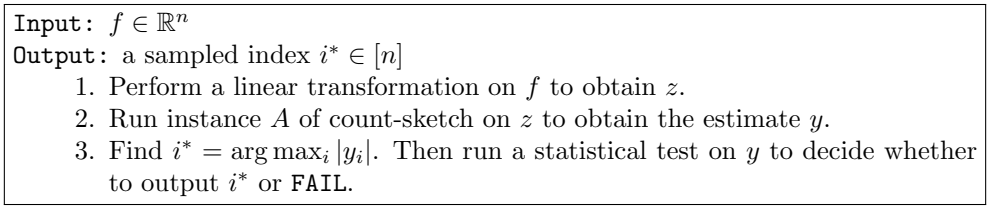

FIG. 2. *Algorithmic template for L_p sampling.*

$g(x)$. In our setting, $g(x) = |x_i|^p/\|x\|_p^p$ could be the probability that an exact L_p sampler should output a fixed coordinate $i \in [n]$ on input frequency vector $x = f \in \mathbb{R}^n$, and the corresponding goal of a private sampler would be to output i with probability $R = (1 \pm \nu)g(x)$ without reveling any more information about x than the value $g(x)$. In this setting, a perfect L_p sampler will allow for a simulation which has $1/\text{poly}(n)$ variational distance to the true distribution of R resulting from the sampler (just by adding small $1/\text{poly}(n)$-sized noise), whereas an approximate $(1 + \nu)$ relative error sampler would only allow for a simulation that is correct up to variational distance ν. This results in approximate samplers being substantially less private than perfect samplers.

1.3. Our techniques. Our main algorithm is inspired by the precision sampling technique used in prior works [AKO10, JST11], but with some marked differences. To describe how our sampler achieves the improvements mentioned above, we begin by observing that all L_p sampling algorithms since [AKO10] have adhered to the same algorithmic template (shown in Figure 2). This template employs the classic count-sketch algorithm of [CCFC02a] as a subroutine, which is easily introduced. For $k \in \mathbb{N}$, let $[k]$ denote the set $\{1, 2, \ldots, k\}$. Given a precision parameter η, count-sketch selects pairwise independent hash functions $h_j : [n] \to [6/\eta^2]$ and $g_j : [n] \to \{1, -1\}$ for $j = 1, 2, \ldots, d$, where $d = \Theta(\log(n))$. Then for all $i \in [d]$, $j \in [6/\eta^2]$, it computes the following linear function $A_{i,j} = \sum_{k \in [n], h_i(k) = j} g_i(k) f_k$ and outputs an approximation y of f given by $y_k = \text{median}_{i \in [d]}\{g_i(k) A_{i,h_i(k)}\}$. We will discuss the estimation guarantee of count-sketch at a later point.

The algorithmic template is as follows. First, perform some linear transformation on the input vector f to obtain a new vector z. Next, run an instance A of count-sketch on z to obtain the estimate y. Finally, run some statistical test on y. If the test fails, then output FAIL; otherwise output the index of the largest coordinate (in magnitude) of y. We first describe how the sampler of [JST11] implements the steps in this template. Afterward we describe the different implementation decisions made in our algorithm that allow it to overcome the limitations of prior approaches.

Prior algorithms. The samplers of [JST11, AKO10] utilize the technique known as *precision sampling*, which employs the following linear transformation. The algorithms first generate random variables $(t_1, \ldots, t_n)$ with limited independence, where each $t_i \sim \text{Uniform}[0, 1]$. Next, each coordinate f_i is scaled by the coefficient $1/t_i^{1/p}$ to obtain the transformed vector $z \in \mathbb{R}^n$ given by $z_i = f_i/t_i^{1/p}$, thus completing step 1 of Figure 2. For simplicity, we now restrict to the case of $p = 1$ and the algorithm of [JST11]. The goal of the algorithm is then to return an item z_i that crosses the threshold $|z_i| > \nu^{-1} R$, where $R = \Theta(\|f\|_1)$ is a constant factor approximation of the L_1. Note that the probability that this occurs is proportional to $\nu |f_i|/\|f\|_1$.

Next, implementing the second step of Figure 2, the vector z is hashed into count-sketch to find an item that has crossed the threshold. Using the stronger *tail-guarantee* of count-sketch, the estimate vector y satisfies $\|y - z\|_\infty \le \sqrt{\eta}\|z_{\text{tail}(1/\eta)}\|_2$,

where $z_{\text{tail}(1/\eta)}$ is z with the $1/\eta$ largest coordinates (in magnitude) set to 0. Now the algorithm runs into trouble when it incorrectly identifies z_i as crossing the threshold when it has not, or vice-versa. However, if the tail error $\sqrt{\eta}\|z_{\text{tail}(1/\eta)}\|_2$ is at most $O(\|f\|_1)$, then since t_i is a uniform variable the probability that z_i is close enough to the threshold to be misidentified is $O(\nu)$, which results in at most $(1 \pm \nu)$ relative error in the sampling probabilities. Thus it will suffice to have $\sqrt{\eta}\|z_{\text{tail}(1/\eta)}\|_2 = O(\|f\|_1)$ with probability $1 - \nu$. To show that this is the case, consider the level sets $I_k = \{z_i \mid z_i \in (\frac{\|f\|_p}{2^{(k+1)/p}}, \frac{\|f\|_p}{2^{k/p}})\}$. One can show that $\mathbb{E}[|I_k|] = 2^k$. We observe here that results of [JST11] can be partially attributed to the fact that for $p < 2$, the total contribution $\Theta(\frac{\|f\|_p^2}{2^{2k/p}}|I_k|)$ of the level sets to $\|z\|_2^2$ decreases geometrically with k, and so with constant probability we have $\|z\|_2 = O(\|f\|_p)$. Moreover, if one removes the top $\log(1/\nu)$ largest items, the contribution of the remaining items to the L_2 is $O(\|f\|_1)$ with probability $1 - \nu$. So taking $\eta = \log(1/\nu)$, the tail error from count-sketch has the desired size. Since the tail error does not include the $1/\eta$ largest coordinates, this holds even conditioned on a fixed value t_{i^*} of the maximizer.

Now with probability ν the guarantee on the error from the prior paragraph does not hold, and in this case one *cannot* still output an index i, as this would result in a ν-*additive* error sampler. Thus, as in step 3 of Figure 2, the algorithm must implement a statistical test to check that the guarantee holds. To do this, using the values of the largest $1/\eta$ coordinates of y, they produce an estimate of the tail-error and output `FAIL` if it is too large. Otherwise, the item $i^* = \arg\max_i |y_i|$ is outputted if $|y_{i^*}| > \nu^{-1}R$. The whole algorithm is run $O(\nu^{-1}\log(1/\delta))$ times so that an index is outputted with probability $1 - \delta$.

Our algorithm. Our first observation is that, in order to obtain a truly perfect sampler, one needs to use different scaling variables t_i. Notice that the approach of scaling by inverse uniform variables and returning a coordinate which reaches a certain threshold T faces the obvious issue of what to return when more than one of the variables $|z_i|$ crosses T. This is solved by simply outputting the maximum of all such coordinates. However, the probability of an index becoming the maximum *and* reaching a threshold is drawn from an entirely different distribution, and for uniform variables t_i this distribution does not appear to be the correct one. To overcome this, we must use a distribution where the maximum index i of the variables $(|f_1 t_2^{-1/p}|, |f_2 t_2^{-1/p}|, \ldots, |f_n t_n^{-1/p}|)$ is drawn *exactly* according to the L_p distribution $|f_i|^p/\|f\|_p^p$. We observe that the distribution of exponential random variables has precisely this property, and thus to implement step 1 of Figure 2 we set $z_i = f_i/t_i^{1/p}$, where t_i is an exponential random variable. We remark that exponential variables have been used in the past, such as for F_p moment estimation, $p > 2$, in [AKO10] and regression in [WZ13]. However, it appears that their applicability to sampling has never before been exploited.

Next, we carry out the count-sketch step by hashing our vector z into a count-sketch data structure A. Because we are only interested in the maximizer of z, we develop a modified version of count-sketch, called *count-max*. Instead of producing an estimate y such that $\|y - z\|_\infty$ is small, count-max simply checks, for each $i \in [n]$, how many times z_i hashed into the largest bucket (in absolute value) of a row of A. If this number is at least a 4/5-fraction of the total number of rows, count-max declares that z_i is the maximizer of z. We show that with high probability, count-max never incorrectly declares an item to be the maximizer, and moreover if $z_i > 20(\sum_{j \neq i} z_j^2)^{1/2}$, then count-max will declare i to be the maximizer. Using the *min-stability* property of exponential random variables, we can show that the maximum item $|z_{i^*}| = \max\{|z_i|\}$

is distributed as $\|f\|_p/E^{1/p}$, where E is another exponential random variable. Thus $|z_{i^*}| = \Omega(\|f\|_p)$ with constant probability. Using a more general analysis of the L_2 norm of the level sets I_k, we can show that $(\sum_{j\neq i^*} z_j^2)^{1/2} = O(\|f\|_p)$. If all these events occur together (with sufficiently large constants), count-max will correctly determine the coordinate $i^* = \arg\max_i\{|z_i|\}$. However, just as in [JST11], we cannot output an index anyway if these conditions do not hold, so we will need to run a statistical test to ensure that they do.

The statistical test. To implement step 3 of the template, our algorithm tests whether count-max declares any coordinate $i \in [n]$ to be the maximizer, and we output `FAIL` if no coordinate is declared as the maximizer. This approach guarantees that we correctly output the maximizer conditioned on not failing. The primary technical challenge will be to show that, conditioned on $i = \arg\max_j\{|z_j|\}$ for some i, the probability of failing the statistical test *does not depend on* i. In other words, conditioning on $|z_i|$ being the maximum does not change the failure probability. Let $z_{D(k)}$ be the kth order statistic of z (i.e., $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(n)}|$). Here the $D(k)$'s are known as *antiranks* of the distribution $(z_1, \ldots, z_n)$. To analyze the conditional dependence, we must first obtain a closed form for $z_{D(k)}$ which separates the dependencies on k and $D(k)$. Hypothetically, if $z_{D(k)}$ depended only on k, then our statistical test would be completely independent of $D(1)$, in which case we could safely fail whenever such an event occurred. Of course, in reality this is not the case. Consider the vector $f = (100n, 1, 1, 1, \ldots, 1) \in \mathbb{R}^n$ and $p = 1$. Clearly we expect z_1 to be the maximizer, and moreover we expect a gap of $\Theta(n)$ between z_1 and $z_{D(2)}$. On the other hand, if you were told that $D(1) \neq 1$, it is tempting to think that $z_{D(1)}$ just *barely* beat out z_1 for its spot as the max, and so z_1 would not be far behind. Indeed, this intuition would be correct, and one can show that the probability that $z_{D(1)} - z_{D(2)} > n$ conditioned on $D(1) = i$ changes by an additive constant depending on whether or not $i = 1$. Conditioned on this gap being smaller or larger, we are more or less likely (respectively) to output `FAIL`. In this setting, the probability of conditional failure can change by an $\Omega(1)$ factor depending on the value of $D(1)$.

To handle scenarios of this form, our algorithm will utilize an additional linear transformation in step 1 of the template. Instead of only scaling by the random coefficients $1/t_i^{1/p}$, our algorithm first *duplicates* the coordinates f_i to remove all heavy items from the stream. If f is the vector from the example above and F is the duplicated vector, then after $\text{poly}(n)$ duplications all copies of the heavy item f_1 will have weight at most $|f_1|/\|F\|_1 < 1/\text{poly}(n)$. By uniformizing the relative weight of the coordinates, this washes out the dependency of $|z_{D(2)}|$ on $D(1)$, since $\|F_{-D(1)}\|_p^p = (1 \pm n^{-\Omega(c)})\|F_{-j}\|_p^p$ after n^c duplications for any $j \in [n^c]$. Notice that this transformation blows up the dimension of f by a $\text{poly}(n)$ factor. However, since our space usage is always $\text{poly}\log(n)$, the result is only a constant factor increase in the complexity.

After duplication, we scale F by the coefficients $1/t_i^{1/p}$, and the rest of the algorithm proceeds as described above. Using expressions for the order statistics $z_{D(k)}$ which separate the dependence into the antiranks $D(j)$ and a set of exponentials $E_1, E_2, \ldots E_n$ *independent* of the antiranks, after duplication we can derive tight concentration of the $z_{D(k)}$'s conditioned on fixed values of the E_i's. Using this concentration result, we decompose our count-max data structure A into two component variables: one independent of the antiranks (the independent component) and a small adversarial noise of relative weight n^{-c}. In order to bound the effect of the adversarial noise on the outcome of our tests we must (1) randomize the threshold for our failure

condition and (2) demonstrate the anticoncentration of the resulting distribution over the independent components of A. This will demonstrate that with high probability, the result of the statistical test is completely determined by the value of the independent component, which allows us to fail without affecting the conditional probability of outputting $i \in [n]$.

Derandomization. Now the correctness of our sampler crucially relies on the full independence of the t_i's to show that the variable $D(1)$ is drawn from precisely the correct distribution (namely, the L_p distribution $|f_i|^p/\|f\|_p^p$). This being the case, we cannot directly implement our algorithm using any method of limited independence. In order to derandomize the algorithm from requiring full-independence, we will use a combination of Nisan's PRG [Nis92], as well as an extension of the recent PRG of [GKM18], which fools certain classes of *Fourier transforms*. We first use a closer analysis of the seed length Nisan's generator required to fool the randomness required for the count-max data structure, which avoids the standard $O(\log n)$-space blow-up, which would be incurred by using Nisan's as a black-box. Once the count-max has been derandomized, we demonstrate how the PRG of [GKM18] can be used to fool *arbitrary* functions of d halfspaces, as long as each of the half-spaces can be specified by a normal vector with bounded bit-complexity. Specifically, we require that each coordinate v_i of the normal vector $v \in \mathbb{R}^m$ that specifies an m-dimensional halfspace has bounded bit complexity; for our application, each coordinate v_i is specified using $O(\log n)$ bits. We use this result to derandomize the exponential variables t_i with a seed of length $O(\log^2(n)(\log\log n)^2)$, which will allow for the total derandomization of our algorithm for $\delta = \Theta(1)$ and $p < 2$ in the same space.

Our derandomization technique is in fact fairly general and can be applied to streaming algorithms beyond the sampler in this work. Namely, we demonstrate that *any* streaming algorithm which stores a linear sketch $A \cdot f$, where the entries of A are independent and can be sampled from with $O(\log(n))$-bits, can be derandomized with only a $O((\log\log n)^2)$-factor increase in the space requirements (see Theorem 5). This improves the $O(\log(n))$–blow-up incurred from black-box usage of Nisan's PRG. As an application, we derandomize the count-sketch variant of Minton and Price [MP14] to use $O(\epsilon^{-2}\log^2(n)(\log\log n)^2)$-bits of space, which gives improved concentration results for count-sketch when the hash functions are fully independent. The problem of improving the derandomization of [MP14] beyond the black-box application of Nisan's PRG was an open problem. We remark that using $O(1/\epsilon^2\log^3(n))$-bits of space in the classic count sketch of [CCFC02a] has strictly better error guarantees than those obtained from derandomizing [MP14] with Nisan's PRG to run in the same space. Our derandomization, in contrast, demonstrates a strong improvement on this, obtaining the same bounds with an $O((\log\log n)^2)$ instead of an $O(\log(n))$ factor blow-up.

Case of $p = 2$. Recalling $p < 2$, we could show that the L_2 norm of the level sets I_k decays geometrically with k. More precisely, for any $\gamma \geq 1$ we have $\|z_{\mathrm{tail}(\gamma)}\|_2 = O(\|F\|_p\gamma^{-1/p+1/2})$ with probability $1 - O(e^{-\gamma})$. Using this, we actually do not need the tight concentration of the $z_{D(k)}$'s, since we can show that the top $n^{c/10}$ coordinates change by at most $(1 \pm n^{-\Omega(c)})$ depending on $D(1)$, and the L_2 norm of the remaining coordinates is only an $O(n^{-c/10(1/p-1/2)})$ fraction of the whole L_2 and can thus be absorbed into the adversarial noise. For $p = 2$, however, each level set I_k contributes weight $O(\|F\|_p^2)$ to $\|z\|_2^2$, so $\|z_{\mathrm{tail}(\gamma)}\|_2 = O(\sqrt{\log n}\|F\|_p)$ even for $\gamma = \mathrm{poly}(n)$. Therefore, for $p = 2$ it is essential that we show concentration of the $z_{D(k)}$'s for *nearly all* k. Since $\|z\|_2^2$ will now be larger than $\|F\|_2^2$ by a factor of $\log n$ with high probability, count-max will only succeed in outputting the largest co-

ordinate when it is an $O(\sqrt{\log n})$ factor larger than expected. This event occurs with probability $1/\log n$, so we will need to run the algorithm $\log n$ times in parallel to get constant probability for a total $O(\log^3 n)$-bits of space. Using the same $O(\log^3 n)$-bit Nisan PRG seed for all $O(\log n)$ repititions, we show that the entire algorithm for $p = 2$ can be derandomized to run in $O(\log^3 n \log 1/\delta)$-bits of space.

Optimizing the runtime. In addition to our core sampling algorithm, we show how the linear transformation step to construct z can be implemented via a parameterized rounding scheme to improve the update time of the algorithm without affecting the space complexity, giving a run-time/relative sampling error trade-off. By rounding the scaling variables $1/t_i^{1/p}$ to powers of $(1+\nu)$, we discretize their support to have size $O(\nu^{-1}\log n)$. We then simulate the update procedure by sampling from the distribution over updates to our count-max data-structure A of duplicating an update and hashing each duplicate independently into A. Our simulation utilizes results on efficient generation of binomial random variables, through which we can iteratively reconstruct the updates to A bin-by-bin instead of duplicate-by-duplicate. In addition, by using an auxiliary heavy hitter data structure, we can improve our query time from the naïve $O(n)$ to $O(\text{poly}\log n)$ without increasing the space complexity.

Estimating the frequency. We show that allowing an additional additive $O(\min\{\epsilon^{-2}, \epsilon^{-p}\log(\frac{1}{\delta_2})\}\log n \log \delta_2^{-1})$ bits of space, we can provide an estimate $\tilde{f} = (1\pm\epsilon)f_i$ of the outputted frequency f_i with probability $1-\delta_2$ when $p < 2$. To achieve this, we use our more general analysis of the contribution of the level sets I_k to $\|z\|_2$ and give concentration bounds on the tail error when the top ϵ^{-p} items are removed. When $p = 2$, for similar reasons as described in the sampling algorithm, we require another $O(\log n)$ factor in the space complexity to obtain a $(1\pm\epsilon)$ estimate. Finally, we demonstrate an $\Omega(\epsilon^{-p}\log n \log \delta_2^{-1})$ lower bound for this problem, which is nearly tight when $p < 2$. To do so, we adapt a communication problem introduced in [JW13], known as *augmented-indexing on large domains*. We weaken the problem so that it need only succeed with constant probability and then show that the same lower bound still holds. Using a reduction from this problem, we show that our lower bound for L_p samplers holds even if the output index is from a distribution with constant *additive* error from the true L_p distribution $|f_i|^p/\|f\|_p^p$.

2. Preliminaries. For $a, b, \epsilon \in \mathbb{R}$, we write $a = b \pm \epsilon$ to denote the containment $a \in [b-\epsilon, b+\epsilon]$. For positive integer n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and $\tilde{O}(\cdot)$ notation to hide $\log(n)$ terms. For any vector $v \in \mathbb{R}^n$, we write $v_{(k)}$ to denote the kth largest coordinate of v in absolute value. In other words, $|v_{(1)}| \geq |v_{(2)}| \geq \cdots \geq |v_{(n)}|$. For any $\gamma \in [n]$, we define $v_{\text{tail}(\gamma)}$ to be v but with the top γ coordinates (in absolute value) set equal to 0. For any $i \in [n]$, we define v_{-i} to be v with the i th coordinate set to 0. We write $|v|$ to denote the entrywise absolute value of v, so $|v|_j = |v_j|$ for all $j \in [n]$. All space bounds stated will be in bits. For our runtime complexity, we assume the unit cost RAM model, where a word of $O(\log(n))$-bits can be operated on in constant time, where n is the dimension of the input streaming vector. Finally, we will use $\tilde{O}$ notation to hide polylog(n) factors; in other words $O(\log^c(n)) = \tilde{O}(1)$ for any constant c.

Formally, a data stream is given by an underlying vector $f \in \mathbb{R}^n$, called the *frequency vector*, which is initialized to 0^n. The frequency vector then receives a stream of m updates of the form $(i_t, \Delta_t) \in [n] \times \{-M, \ldots, M\}$ for some $M > 0$ and $t \in [m]$. The update (i, Δ) causes the change $f_{i_t} \leftarrow f_{i_t} + \Delta_t$. For simplicity, we make the common assumption [BCIW16] that $\log(mM) = O(\log(n))$, though our results generalize naturally to arbitrary n, m. Since the updates (i_t, Δ_t) can cause coordinates

f_i to become negative, this is known as the *general turnstile* model of streaming, as opposed to the sometimes considered *strict turnstile* model, which forces $f_i \geq 0$ at all time steps. We remark that both of these models are more general than the *insertion only* model, which restricts that $\Delta_t \geq 0$ for all t. In this work, we consider only the most general model, namely, the general turnstile model.

In this paper, we will need Khintchine's and McDiarmid's inequality.

FACT 1 (Khintchine inequality [Haa81]). *Let $x \in \mathbb{R}^n$ and $Q = \sum_{i=1}^n \varphi_i x_i$ for i.i.d. random variables φ_i uniform on $\{1, -1\}$. Then $\mathbf{Pr}[|Q| > t\|x\|_2] < 2e^{-t^2/2}$.*

FACT 2 (McDiarmid's inequality [McD89]). *Let $X_1, X_2, \ldots, X_n$ be independent random variables, and let $\psi(x_1, \ldots, x_n)$ by any function that satisfies*

$$\sup_{x_1,\ldots,x_n,\hat{x}_i} \left|\psi(x_1, x_2, \ldots, x_n) - \psi(x_1, \ldots, x_{i-1}, \hat{x}_i, x_{i+1}, \ldots, x_n)\right| \leq c_i \quad \text{for } 1 \leq i \leq n.$$

Then for any $\epsilon > 0$, we have

$$\mathbf{Pr}\Big[\Big|\psi(X_1, \ldots, X_n) - \mathbb{E}\Big[\psi(X_1, \ldots, X_n)\Big]\Big| \geq \epsilon\Big] \leq 2\exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}\right).$$

Our analysis will use stability properties of Gaussian random variables.

DEFINITION 2. *A distribution $\mathcal{D}_p$ is said to be p-stable if whenever $X_1, \ldots, X_n \sim \mathcal{D}_p$ are drawn independently, we have*

$$\sum_{i=1}^n a_i X_i = \|a\|_p X$$

for any fixed vector $a \in \mathbb{R}^n$, where $X \sim \mathcal{D}_p$ is drawn from the same distribution. In particular, the Gaussian random variables $\mathcal{N}(0, 1)$ are p-stable for $p = 2$ (i.e., $\sum_i a_i g_i = g \cdot \|a\|_2$, where $g, g_1, \ldots, g_n$ are Gaussian).

2.1. Count-sketch and count-max. Our sampling algorithm will utilize a modification of the well-known data structure known as count-sketch (see [CCFC02a] for further details). We now introduce the description of count-sketch which we will use for the remainder of the paper. The count-sketch data structure is a table A with d rows and k columns. When run on a stream $f \in \mathbb{R}^n$, for each row $i \in [d]$, count-sketch picks a uniform random mapping $h_i : [n] \to [k]$ and $g_i : [n] \to \{1, -1\}$. Generally, h_i and g_i need only be 4-wise independent hash functions, but in this paper we will use fully independent hash functions (and later relax this condition when derandomizing). Whenever an update Δ to item $v \in [n]$ occurs, count-sketch performs the following updates:

$$A_{i,h_i(v)} \leftarrow A_{i,h_i(v)} + \Delta g_i(v) \quad \text{for } i = 1, 2, \ldots, d.$$

Note that while we will not implement the h_i's as explicit hash functions, and instead generate i.i.d. random variables $h_i(1), \ldots, h_i(n)$, we will still use the terminology of hash functions. In other words, by *hashing* the update (v, Δ) into the row A_i of count-sketch, we mean that we are updating $A_{i,h_i(v)}$ by $\Delta g_i(v)$. By hashing the coordinate f_v into A, we mean updating $A_{i,h_i(v)}$ by $g_i(v) f_v$ for each $i = 1, 2, \ldots, d$. Using this terminology, each row of count-sketch corresponds to randomly hashing the indices in $[n]$ into k buckets, and then each bucket in the row is a sum of the frequencies f_i of the items which are hashed to it multiplied by random ± 1 signs. In general, count-sketch is used to obtain an estimate vector $y \in \mathbb{R}^n$ such that $\|y - f\|_\infty$ is small. This vector y satisfies the following guarantee.

THEOREM 1. *If $d = \Theta(\log(1/\delta))$ and $k = 6/\epsilon^2$, then for a fixed $i \in [n]$ we have $|y_i - f_i| < \epsilon \|f_{\mathrm{tail}(1/\epsilon^2)}\|_2$ with probability $1 - \delta$, where y is given*

$$y_j = \mathrm{median}_{i \in [d]} A_{i,h_i(j)}(g_i(j))^{-1} \text{ for all } j \in [n].$$

Moreover, if $c \geq 1$ is any constant, and we set $d = \Theta(\log n)$, then we have $\|y - f\|_\infty < \epsilon \|f_{\mathrm{tail}(1/\epsilon^2)}\|_2$ with probability $1 - n^{-c}$. Furthermore, if we instead set $y_j = \mathrm{median}_{i \in [d]} |A_{i,h_i(j)}|$, then the same two bounds above hold replacing f with $|f|$.

In this work, however, we are only interested in determining the index of the heaviest item in f, that is, $i^* = \arg\max_i |f_i|$. So we utilize a simpler estimation algorithm based on the count-sketch data structure that tests whether a fixed $j \in [n]$ if $j = \arg\max_i |f_i|$. For analysis purposes, instead of having the g_i's be random signs, we draw $g_i(v) \sim \mathcal{N}(0,1)$ as i.i.d. Gaussian variables. Then for a fixed j, set $\alpha_j = \left|\{i \in [d] \mid |A_{i,h_i(j)}| = \max_{r \in [k]} |A_{i,r}|\}\right|$, and we declare $j = i^*$ to be the maximizer if $\alpha_j > \frac{4}{5}d$. The algorithm computes α_j for all $j \in [n]$ and outputs the first index j that satisfies $\alpha_j > \frac{4}{5}d$ (there will only be one with high probability). To distinguish this modified querying protocol from the classic count-sketch, we refer to this algorithm as count-max. To refer to the data structure A itself, we will use the terms count-sketch and count-max interchangeably.

We will prove our result for the guarantee of count-max in the presence of the following generalization. Before computing the values of α and reporting a maximizer as above, we will scale each bucket $A_{i,j}$ of count-max by a uniform random variable $\mu_{i,j} \sim \mathtt{Uniform}(\frac{99}{100}, \frac{101}{100})$. This generalization will be used for technical reasons in our analysis of Lemma 3. Namely, we will need it to ensure that our failure threshold of our algorithm is randomized, which will allow us to handle small adversarial errors.

LEMMA 1. *Let $c \geq 1$ be an arbitrarily large constant, set $d = \Theta(\log(n))$ and $k = 2$, and let A be a $d \times k$ instance of count-max run on $f \in \mathbb{R}^n$ using fully independent hash functions h_i and Gaussian random variables $g_i \sim \mathcal{N}(0,1)$. Then with probability $1 - n^{-c}$ the following holds: for every $i \in [n]$, if $|f_i| > 20\|f_{-i}\|_2$, then count-max declares i to be the maximum, and if $|f_i| \leq \max_{j \in [n] \setminus \{i\}} |f_j|$, then count-max does not declare i to be the maximum. Thus if count-max declares $|f_i|$ to be the largest coordinate of f, it will be correct with high probability. Moreover, this result still holds if each bucket $A_{i,j}$ is scaled by a $\mu_{i,j} \sim \mathtt{Uniform}(\frac{99}{100}, \frac{101}{100})$ before reporting.*

Proof. First suppose $|f_i| > 20\|f_{-i}\|_2$, and consider a fixed row j of A. Without loss of generality (WLOG) i hashes to $A_{j,1}$, and thus using the 2-stability of Gaussians (Definition 2), we have $A_{j,1} = \mu_{j,1} g^1 \|f^1\|_2$ and $A_{j,2} = \mu_{j,2} g^2 \|f^2\|_2$, where f^k is f restricted to the coordinates that hash to bucket $A_{j,k}$, and $g^1, g^2 \sim \mathcal{N}(0,1)$. Since $|f^1\|_2 > 20|f^2\|_2$ and $\mu_{i,j} \sim \mathtt{Uniform}(\frac{99}{100}, \frac{101}{100})$, the probability that $|A_{j,2}| > |A_{j,1}|$ is less than the probability that one $\mathcal{N}(0,1)$ Gaussian is 19 times larger than another, which can be bounded by 15/100 by direct computation. Thus i hashes into the max bucket in a row of A with probability at least 85/100, so by Chernoff bounds, taking $d = \Omega(c \log(n))$, with probability $1 - n^{-2c}$ we have that f_i is in the largest bucket at least a 4/5 fraction of the time, which completes the first claim.

Now suppose i is not a unique max, and let i^* be such that $|f_{i^*}|$ is maximal. Then conditioned on i, i^* not hashing to the same bucket, the probability that f_i hashes to a larger bucket than f_{i^*} is at most 1/2. To see this, note that conditioned on this, one bucket is distributed as $\mu(g_j(i^*) f_{i^*} + G)$ and the other as $\mu'(g_j(i) f_i + G')$, where G, G' μ, μ', and $g_j(i^*) f_{i^*}, g_j(i) f_i$ are identically distributed random variables. Thus the probability that f_i is in the maximal bucket is at most 3/4, and so by

Chernoff bounds f_i will hash to strictly less than $(4d/5)$ of the maximal buckets with probability $1 - n^{-2c}$. Union bounding over all $j \in [n]$ gives the desired result. ∎

COROLLARY 1. *In the setting of Lemma* 1*, with probability* $1-O(n^{-c})$*, count-max will never report an index* $i \in [n]$ *as being the maximum if* $|f_i| < \frac{1}{100}\|f\|_2$.

Proof. Suppose $|f_i| < \frac{1}{100}\|f\|_2$, and in a given row WLOG i hashes to $A_{j,1}$. Let f_{-i} be f with the coordinate i set equal to 0. Then we have $A_{j,1} = \mu_{j,1}g^1\|f^1\|_2$ and $A_{j,2} = \mu_{j,2}g^2\|f^2\|_2$, where f^k is f restricted to the coordinates that hash to bucket $A_{j,k}$, and $g^1, g^2 \sim \mathcal{N}(0,1)$. Since f^1_{-i} and $f^2_{-i} = f^2$ are identically distributed, with probability $1/2$ we have $\|f^2_{-i}\|_2 \geq \|f^1_{-i}\|_2$. Conditioned on this, we have $\|f^2_{-i}\|_2^2 \geq \|f^1\|_2^2 - |f^2_i|^2 \geq \|f^1\|_2^2 - \|f^2_{-i}\|_2^2/50$, so $\|f^2_{-i}\|_2(1+1/50)^{1/2} \geq \|f^1\|_2^2$. So conditioned on $\|f^2_{-i}\|_2 > \|f^1_{-i}\|_2$, we have $|A_{j,1}| < |A_{j,2}|$ whenever one Gaussian is $(101/100)(1+1/50)^{1/2}$ times larger than another in magnitude, which occurs with probability greater than $1/2 - 1/25$. So i hashes into the max bucket with probability at most $1/2 + 1/2(1/2 + 1/25) = 77/100$, and thus by Chernoff bounds, taking c sufficiently large and union bounding over all $i \in [n]$, i will hash into the max bucket at most a $79/1000 < 4/5$ fraction of the time with probability $1-O(n^{-c})$, as needed. ∎

3. Exponential order statistics. In this section, we discuss several useful properties of the order statistics of n independent nonidentically distributed exponential random variables. Let $(t_1, \ldots, t_n)$ be independent exponential random variables where t_i has mean $1/\lambda_i$ (equivalently, t_i has rate λ_i). Recall that t_i is given by the cumulative distribution function $\mathbf{Pr}[t_i < x] = 1 - e^{-\lambda_i x}$. Our main L_p sampling algorithm will require a careful analysis of the distribution of values $(t_1, \ldots, t_n)$, which we will now describe. We begin by noting that constant factor scalings of an exponential variable result in another exponential variable.

FACT 3 (scaling of exponentials). *Let* t *be exponentially distributed with rate* λ*, and let* $\alpha > 0$*. Then* αt *is exponentially distributed with rate* λ/α

Proof. The cumulative distribution function (cdf) of αt is given by $\mathbf{Pr}[t < x/\alpha] = 1 - e^{-\lambda x/\alpha}$, which is the cdf of an exponential with rate λ/α. ∎

We would now like to study the order statistics of the variables $(t_1, \ldots, t_n)$, where t_i has rate λ_i. To do so, we introduce the *antirank vector* $(D(1), D(2), \ldots, D(n))$, where for $k \in [n]$, $D(k) \in [n]$ is a random variable which gives the index of the kth smallest exponential.

DEFINITION 3. *Let* $(t_1, \ldots, t_n)$ *be independent exponentials. For* $k = 1, 2, \ldots, n$*, we define the* k*th antirank* $D(k) \in [n]$ *of* $(t_1, \ldots, t_n)$ *to be the values* $D(k)$ *such that* $t_{D(1)} \leq t_{D(2)} \leq \cdots \leq t_{D(n)}$.

Using the structure of the antirank vector, it has been observed [Nag06] that there is a simple form for describing the distribution of $t_{D(k)}$ as a function of $(\lambda_1, \ldots, \lambda_n)$ and the antirank vector.

FACT 4 ([Nag06]). *Let* $(t_1, \ldots, t_n)$ *be independently distributed exponentials, where* t_i *has rate* $\lambda_i > 0$*. Then for any* $k = 1, 2, \ldots, n$*, we have*

$$t_{D(k)} = \sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{n} \lambda_{D(j)}},$$

where the $E_1, E_2, \ldots, E_n$*'s are i.i.d. exponential variables with mean* 1 *and are independent of the antirank vector* $(D(1), D(2), \ldots, D(n))$.

FACT 5 ([Nag06]). *For any $i = 1, 2, \ldots, n$, we have*

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

We now describe how these properties will be useful to our sampler. Let $f \in \mathbb{R}^n$ be any vector presented in a general turnstile stream (note that f must have integral valued coordinates by the definition of the model). We can generate i.i.d. exponentials $(t_1, \ldots, t_n)$, each with rate 1, and construct the random variable $z_i = f_i/t_i^{1/p}$, which can be obtained in a stream by scaling updates to f_i by $1/t_i^{1/p}$ as they arrive. By Fact 3, the variable $|z_i|^{-p} = t_i/|f_i|^p$ is exponentially distributed with rate $\lambda_i = |f_i|^p$. Now let $(D(1), \ldots, D(n))$ be the antirank vector of the exponentials $(t_1/|f_1|^p, \ldots, t_n/|f_n|^{1/p})$. By Fact 5, we have $\mathbf{Pr}[D(1) = i] = \mathbf{Pr}[i = \arg\min\{|z_1|^{-p}, \ldots, |z_n|^{-p}\}] = \mathbf{Pr}[i = \arg\max\{|z_1|, \ldots, |z_n|\}] = \frac{\lambda_i}{\sum_j \lambda_j} = \frac{|f_i|^p}{\|f\|_p^p}$. In other words, the probability that $|z_i| = \arg\max_j\{|z_j|\}$ is precisely $|f_i|^p/\|f\|_p^p$, so for a perfect L_p sampler it suffices to return $i \in [n]$ with $|z_i|$ maximum. Now note $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(n)}|$, and in this scenario the statement of Fact 4 becomes

$$z_{D(k)} = \left(\sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{N} \lambda_{D(j)}} \right)^{-1/p} = \left(\sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{N} f_{D(j)}^p} \right)^{-1/p},$$

where E_i's are i.i.d. exponential random variables with mean 1 and are independent of the antirank vector $(D(1), \ldots, D(n))$. We call the exponentials E_i the *hidden exponentials*, as they do not appear in the actual execution of the algorithm, and will be needed for analysis purposes only.

4. The sampling algorithm. We now provide intuition for the workings of our main sampling algorithm. Our algorithm scales the input stream by inverse exponentials to obtain a new vector z. Namely, we scale f_i by $1/t_i^{1/p}$ to obtain z_i. Let $D = (D(1), \ldots, D(n))$ be the indices such that $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(n)}|$. Note that D is precisely the antiranks of the variables $(t_1, \ldots, t_n)$, where t_i is exponential with rate $|f_i|^p$, because we have $|z_i|^p = 1/t_i$ and $t_{D(1)} \leq t_{D(2)} \leq \cdots \leq t_{D(n)}$. Due to this correspondence, we will also refer to D as the antirank vector of z. We have seen in the prior section that we can write the order statistics $z_{D(k)}$ as a function of the antirank vector D and the hidden exponentials E_i, which describe the "scale" of the order statistics. Importantly, the hidden exponentials are independent of the antiranks. We would like to determine the index i for which $D(1) = i$, but this may not always be possible. This is the case when the largest element $|z_{D(1)}|$ is not sufficently larger than the remainig L_2 mass $\sum_{j>1}(|z_{D(j)}|^2)^{1/2}$. In such a case, count-max will not declare any index to be the largest, and we would therefore like to output `FAIL`. Note that this event is more likely when there is another element $|z_{D(2)}|$ that is very close to $|z_{D(1)}|$ in size, as whenever the two elements do not collide in count-max, it is less likely that $|z_{D(1)}|$ will be in the max bucket.

Now consider the trivial situation where $f_1 = f_2 = \cdots = f_n$. Here the variables $z_{D(k)}$ have no dependence at all on the antirank vector D. In this case, the condition of failing is independent of $D(1)$, so we can safely fail whenever we cannot determine the maximum index. On the other hand, if the values $|f_i|$ vary wildly, the variables $z_{D(k)}$ will depend highly on the antiranks. In fact, if there exists f_i with $|f_i|^p \geq \epsilon\|f\|_p^p$, then the probability that $|z_{D(1)}| - |z_{D(2)}|$ is above a certain threshold can change by a

L_p `Sampler`

1. Set $d = \Theta(\log n)$, instantiate a $d \times 2$ count-max table A, and set $\mu_{i,j} \sim \texttt{Uniform}[\frac{99}{100}, \frac{101}{100}]$ for each $(i,j) \in [d] \times [2]$.
2. Duplicate updates to f to obtain the vector $F \in \mathbb{R}^{n^c}$ so that $f_i = F_{i_j}$ for all $i \in [n]$ and $j = 1, 2, \ldots, n^{c-1}$ for some fixed constant c.
3. Choose i.i.d. exponential random variables $t = (t_1, t_2, \ldots, t_{n^c})$, and construct the stream $\zeta_i = F_i \cdot \texttt{rnd}_\nu(1/t_i^{1/p})$.
4. Run A on the stream ζ. Upon the end of the stream, set $A_{i,j} \leftarrow \mu_{i,j} A_{i,j}$ for all $(i,j) \in [d] \times [2]$.
5. If count-max declares that an index $i_j \in [n^c]$ is the max for some $j \in [n^{c-1}]$ based on the data structure A, then output $i \in [n]$. If A does not declare any index to be the max, output `FAIL`.

FIG. 3. *Our main L_p sampling algorithm.*

$(1 \pm \epsilon)$ factor conditioned on $D(1) = i$, as opposed to $D(1) = j$ for a smaller $|f_j|$. Given this, the probability that we fail can change by a multiplicative $(1 \pm \epsilon)$ conditioned on $D(1) = i$ as opposed to $D(1) = j$. In this case, we cannot output `FAIL` when count-max does not report a maximizer, lest we suffer a $(1 \pm \epsilon)$ error in outputting an index with the correct probability.

To handle this, we must remove the heavy items from the stream to weaken the dependence of the values $z_{D(k)}$ on the antiranks, which we carry out by duplication of coordinates. For the purpose of efficiency, we carry out the duplication via a rounding scheme which will allow us to generate and quickly hash updates into our data-structures (section 5). We will show that, conditioned on the fixed values of the E_i's, the variables $z_{D(k)}$ are highly concentrated and therefore nearly independent of the antiranks ($z_{D(k)}$ depends only on k and not $D(k)$). By randomizing the failure threshold to be anticoncentrated, the small adversarial dependence of $z_{D(k)}$ on $D(k)$ cannot nontrivially affect the conditional probabilities of failure, leading to small relative error in the resulting output distribution.

The L_p sampler. We now describe our sampling algorithm, as shown in Figure 3. To begin with, we instantiate a $d \times 2$ count-max table A, where $d = \Theta(\log n)$, and generate uniform variables $\mu_{i,j} \sim \texttt{Uniform}[\frac{99}{100}, \frac{101}{100}]$ for each $(i,j) \in [d] \times [2]$ to use as scalings for each bucket in the table. Let $f \in \mathbb{R}^n$ be the input vector of the stream. As the stream arrives, we duplicate updates to each coordinate f_i a total of n^{c-1} times to obtain a new vector $F \in \mathbb{R}^{n^c}$. More precisely, for $i \in [n]$ we set $i_j = (i-1)n^{c-1} + j$ for $j = 1, 2, \ldots, n^{c-1}$, and then we will have $F_{i_j} = f_i$ for all $i \in [n]$ and $j \in [n^{c-1}]$. We then call F_{i_j} a duplicate of f_i. Whenever we use i_j as a subscript in this way it will refer to a duplicate of i, whereas a single subscript i will be used both to index into $[n]$ and $[n^c]$. Note that this duplication has the effect that $|F_i|^p \leq n^{-c+1}\|F\|_p^p$ for all $p > 0$ and $i \in [n^c]$.

We then generate i.i.d. exponential rate 1 random variables $(t_1, \ldots, t_{n^c})$ and define the vector $z \in \mathbb{R}^{n^c}$ by $z_i = F_i/t_i^{1/p}$. As shown in section 3, we have $\mathbf{Pr}[i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}] = |F_{i_j}|^p/\|F\|_p^p$. Since $\sum_{j \in [n^{c-1}]} |F_{i_j}|^p/\|F\|_p^p = |f_i|^p/\|f\|_p^p$, it will therefore suffice to find $i_j \in [n^c]$ for which $i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}$ and return the index $i \in [n]$. The assumption that the t_i's are i.i.d. will later be relaxed in section 5 while derandomizing the algorithm. In section 5, we also demonstrate that all relevant continuous distributions will be made discrete without affecting the perfect sampling guarantee.

Now fix any sufficiently large constant c, and fix $\nu > n^{-c}$. To speed up the update time, instead of explicitly scaling F_i by $1/t_i^{1/p}$ to construct the stream z, our algorithm instead scales F_i by $\texttt{rnd}_\nu(1/t_i^{1/p})$, where $\texttt{rnd}_\nu(x)$ rounds $x > 0$ down to the nearest value in $\{\dots, (1+\nu)^{-1}, 1, (1+\nu), (1+\nu)^2, \dots\}$. In other words, $\texttt{rnd}_\nu(x)$ rounds x down to the nearest power of $(1+\nu)^j$ (for $j \in \mathbb{Z}$). This results in a separate stream $\zeta \in \mathbb{R}^{n^c}$, where $\zeta_i = F_i \cdot \texttt{rnd}_\nu(1/t_i^{1/p})$. Note $\zeta_i = (1 \pm O(\nu))z_i$ for all $i \in [n^c]$. Importantly, note that this rounding is order preserving. Thus, if ζ has a unique largest coordinate $|\zeta_{i^*}|$, then $|z_{i^*}|$ will be the unique largest coordinate of z.

Having constructed the transformed stream ζ, we then run a $d \times 2$ instance $A \in \mathbb{R}^{d\times 2}$ of count-max (from section 2.1), with $d = \Theta(\log(n))$, on ζ. At the end of the stream, we scale each bucket $A_{i,j}$ by a uniform random variable $\mu_{i,j}$ from the interval $[\frac{99}{100}, \frac{101}{100}]$. This step ensures that the failure threshold is randomized, so that a small adversarial error can only affect the output of the algorithm with extremely low probability (see Lemma 3). Now recall that count-max will either declare an index $i_j \in [n^c]$ as being the maximum or report nothing. If an index i_j is returned, where i_j is the jth copy of index $i \in [n]$, then our algorithm outputs the index i. If count-max does not report an index, we return `FAIL`. Let $i^* = \arg\max_i |\zeta_i| = D(1)$ (where $D(1)$ is the first antirank as in section 3). By the guarantee of Lemma 1, we know that if $|\zeta_{i^*}| \geq 20\|\zeta_{-i^*}\|_2$, then with probability $1 - n^{-c}$ count-max will return the index $i^* \in [n^c]$. Moreover, with the same probability, count-max will never return an index that is not the unique maximizer. To prove correctness, therefore, it suffices to analyze the conditional probability of failure given $D(1) = i$. Let $N = |\{i \in [n^c] \mid F_i \neq 0\}|$ (N is the support size of F). We can assume that $N \neq 0$ (to check this one could run, for instance, the $O(\log^2(n))$-bit support sampler of [JST11]). Note that $n^{c-1} \leq N \leq n^c$. The following fact is straightforward.

FACT 6. *For $p \in (0, 2]$, suppose that we choose the constant c such that $mM \leq n^{c/20}$, where note we have $|F_i| \leq mM$ for all $i \in [N]$. Then if $S \subset \{i \in [n^c] \mid F_i \neq 0\}$ is any subset, then $\sum_{i\in S} |F_i|^p \geq \frac{|S|}{N} n^{-c/10}\|F\|_p^p$.*

Proof. We know that $|F_i|^p \leq (mM)^p \leq n^{c/10}$ using $p \leq 2$. Then each nonzero value $|F_i|^p$ is at most an $n^{-c/10}$ fraction of any other item $|F_j|^p$, and in particular of the average item weight. It follows that $|F_i|^p \geq n^{-c/10}\frac{\|F\|_p^p}{N}$ for all $i \in [N]$, which results in the stated fact. ∎

As in section 3, we now use the antirank vector $D(k)$ to denote the index of the kth largest value of z_i in absolute value. In other words, $D(k)$ is the index such that $|z_{D(k)}|$ is the kth largest value in the set $\{|z_1|, |z_2|, \dots, |z_{n^c}|\}$. Note that the $D(k)$'s are also the antiranks of the vector ζ, since rounding z into ζ preserves partial ordering. For the following lemma, it suffices to consider only the exponentials t_i with $F_i \neq 0$, and we thus consider only values of k between 1 and N. Thus $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(N)}|$. Moreover, we have that $|z_{D(k)}|^{-p} = \frac{t_{D(k)}}{|F_{D(k)}|^p}$ is the kth smallest of all the $\frac{t_i}{|F_i|^p}$'s and by the results of section 3 can be written as $|z_{D(k)}|^{-p} = \sum_{\tau=1}^k \frac{E_\tau}{\sum_{j=\tau}^N |F_{D(j)}|^p}$, where the E_τ are i.i.d. exponentials and independent of the antirank vector D. We will make use of this in the following lemma.

LEMMA 2. *For every $1 \leq k < N - n^{9c/10}$, we have*

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

with probability $1 - O(e^{-n^{c/3}})$.

Proof. Let $\tau < N - n^{9c/10}$. We can write $\sum_{j=\tau}^{N} |F_{D(j)}|^p$ as a deterministic function $\psi(t_1, \ldots, t_N)$ of the random scaling exponentials $t_1, \ldots, t_N$ corresponding to $F_i \neq 0$. We first argue that

$$|\psi(t_1, \ldots, t_N) - \psi(t_1 \ldots, t_{i-1}, t_i', t_{i+1}, \ldots, t_N)| < 2 \max_j \{F_j^p\} < 2n^{-c+1}\|F\|_p^p.$$

This can be seen from the fact that changing a value of t_i can only have the effect of adding (or removing) $|F_i|^p$ to the sum $\sum_{j=\tau}^{N} |F_{D(j)}|^p$ and removing (or adding) a different $|F_l|$ from the sum. The resulting change in the sum is at most $2 \max_j \{|F_j|^p\}$, which is at most $2n^{-c+1}\|F\|_p^p$ by duplication. Set $T = N - \tau + 1$. Since the t_i's are independent, we apply McDiarmid's inequality (Fact 2) to obtain

(1)
$$\mathbf{Pr}\left[\left|\sum_{j=\tau}^{N} \left|F_{D(j)}|^p - \mathbb{E}\left[\sum_{j=\tau}^{N} |F_{D(j)}|^p\right]\right| > \epsilon T n^{-c}\|F\|_p^p\right] \leq 2\exp\left(\frac{-2\epsilon^2 T^2 n^{-2c}\|F\|_p^{2p}}{n^c(2n^{-c+1}\|F\|_p^p)^2}\right)$$
$$\leq 2\exp\left(-\frac{1}{2}\epsilon^2 T^2 n^{-c-2}\right).$$

Setting $\epsilon = \Theta(n^{-c/5})$ and using $T > n^{9c/10}$, this is at most $2\exp(-\frac{1}{2}n^{2c/5-2})$. To show concentration up to a $(1 \pm O(n^{-c/10}))$ factor, it remains to show that $\mathbb{E}[\sum_{j=\tau}^{N} |F_{D(j)}|^p] = \Omega(Tn^{-11c/10}\|F\|_p^p)$. This follows from the Fact 6, which gives $\sum_{j=0}^{T} |F_{D(-j)}|^p \geq n^{-c/10}(Tn^{-c}\|F\|_p^p)$ deterministically. Now recall that $|z_{D(k)}| = [\sum_{\tau=1}^{k} \frac{E_\tau}{\sum_{j=\tau}^{N} |F_{D(-j)}|^p}]^{-1/p}$. We have just shown that

$$\sum_{j=\tau}^{N} |F_{D(j)}|^p = (1 \pm O(n^{-c/10}))\mathbb{E}\left[\sum_{j=\tau}^{N} |F_{D(j)}|^p\right],$$

so we can union bound over all $\tau = 1, 2, \ldots, N - n^{9c/10}$ to obtain

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

for all $k \leq N - n^{9c/10}$ with probability

$$1 - O(n^c e^{-n^{2c/5-2}}) = 1 - O(e^{n^{c/3}}).$$ ∎

We use this result to show that our failure condition is nearly independent of the value $D(1)$. Let $\mathcal{E}_1$ be the event that Lemma 2 holds. Let $\neg$FAIL be the event that the algorithm L_p Sampler does not output FAIL.

LEMMA 3. *For $p \in (0, 2]$ a constant bounded away from 0 and any $\nu \geq n^{-c/60}$, $\mathbf{Pr}[\neg\texttt{FAIL} \mid D(1)] = \mathbf{Pr}[\neg\texttt{FAIL}] \pm \tilde{O}(\nu)$ for every possible $D(1) \in [N]$.*

Proof. Let $U_{D(k)} = (\sum_{\tau=1}^{k} \frac{E_\tau}{\mathbb{E}[\sum_{j=\tau}^{N} |F_{D(j)}|^p]})^{-1}$, which is independent of the antirank vector D (in fact, it is totally determined by k and the hidden exponentials E_i). Then by Lemma 2, conditioned on $\mathcal{E}_1$, for every $k < N - n^{9c/10}$ we have $|z_{D(k)}| = U_{D(k)}^{1/p}(1 \pm O(n^{-c/10}))^{1/p} = U_{D(k)}^{1/p}(1 \pm O(\frac{1}{p}n^{-c/10}))$ (using the iden-

tity $(1+x) \le e^x$ and the Taylor expansion of e^x). Then for c sufficiently large, we have $|\zeta_{D(k)}| = U_{D(k)}^{1/p}(1 \pm O(\nu))$, and so for all $p \in (0,2]$ and $k < N - n^{9c/10}$

$$|\zeta_{D(k)}| = U_{D(k)}^{1/p} + U_{D(k)}^{1/p} V_{D(k)},$$

where $V_{D(k)}$ is some random variable that satisfies $|V_{D(k)}| = O(\nu)$. Now consider a bucket $A_{i,j}$ for $(i,j) \in [d] \times [2]$. Let $\sigma_k = \mathtt{sign}(z_k) = \mathtt{sign}(\zeta_k)$ for $k \in [n^c]$. Then we write $A_{i,j}/\mu_{i,j} = \sum_{k \in B_{ij}} \sigma_{D(k)} |\zeta_{D(k)}| g_i(D(k)) + \sum_{k \in S_{ij}} \sigma_{D(k)} |\zeta_{D(k)}| g_i(D(k))$, where $B_{ij} = \{k \le N - n^{9c/10} | h_i(D(k)) = j\}$ and $S_{ij} = \{n^c \ge k > N - n^{9c/10} | h_i(D(k)) = j\}$. Here we define $\{D(N+1), \dots, D(n^c)\}$ to be the set of indices i with $F_i = 0$ (in any ordering, as they contribute nothing to the sum). Also recall that $g_i(D(k)) \sim \mathcal{N}(0,1)$ is the i.i.d. Gaussian coefficent associated to item $D(k)$ in row i of A. So

$$A_{i,j}/\mu_{i,j} = \sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p} + \sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p} V_{D(k)} + \sum_{k \in S_{ij}} g_i(D(k)) \zeta_{D(k)}.$$

Importantly, observe that since the variables $h_i(D(k))$ are fully independent, the sets $B_{i,j}, S_{i,j}$ are independent of the antirank vector D. In other words, the values $h_i(D(k))$ are independent of the values $D(k)$ (and of the entire antirank vector), since $\{h_i(1), \dots, h_i(n^c)\} = \{h_i(D(1)), \dots, h_i(D(n^c))\}$ are i.i.d. Note that this would not necessarily be the case if $\{h_i(1), \dots, h_i(n^c)\}$ were only ℓ-wise independent for some $\ell = o(n^c)$. So we can condition on a fixed set of values $\{h_i(D(1)), \dots, h_i(D(n^c))\}$ now, which fixes the sets $B_{i,j}, S_{i,j}$. Now let $U_{i,j}^* = |\sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p}|$.

CLAIM 1. *For all $i, j, \in [d] \times [2]$ and $p \in (0,2]$, we have*

$$\left| \sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p} V_{D(k)} + \sum_{k \in S_{ij}} g_i(D(k)) \zeta_{D(k)} \right| = O\left(\nu(|A_{i,1}| + |A_{i,2}|)\right)$$

with probability $1 - O(\log(n) n^{-c/60})$.

Proof. By the 2-stability of Gaussians (Definition 2),

$$\left| \sum_{k \in S_{ij}} g_i(D(k)) \zeta_{D(k)} \right| = O(\sqrt{\log(n)}(\sum_{k \in S_{i,j}} (2 z_{D(k)})^2)^{1/2})$$

with probability $1 - n^{-c}$. This is a sum over a subset of the $n^{9c/10}$ smallest items $|z_i|$, and thus $\sum_{k \in S_{i,j}} z_{D(k)}^2 < \frac{n^{9c/10}}{N} \|z\|_2^2$, giving

$$\left| \sum_{k \in S_{ij}} g_i(D(k)) \zeta_{D(k)} \right| = O(\sqrt{\log(n)} n^{-c/30} \|z\|_2).$$

Now WLOG $A_{i,1}$ is such that $\sum_{k \in B_{i,1} \cup S_{i,1}} \zeta_{D(k)}^2 > \frac{1}{2}\|\zeta\|_2^2$. Then $|A_{i,1}| \ge |g| \|z\|_2^2/3$, where $g \sim \mathcal{N}(0,1)$. Via the cdf of a Gaussian, it follows with probability $1 - O(n^{-c/60})$ that $|A_{i,1}| > n^{-c/60} \|z\|_2^2 = \Omega((n^{c/60}/\sqrt{\log(n)})|\sum_{k \in S_{ij}} g_i(D(k)) \zeta_{D(k)}|)$. Scaling ν by a $\log(n)$ factor gives $|\sum_{k \in S_{ij}} g_i(D(k)) \zeta_{D(k)}| = O(\nu |A_{i,1}|)$. Next, using that $|V_{D(k)}| = O(\nu)$, we have $|\sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p} V_{D(k)}| = O(\nu) |\sum_{k \in B_{ij}}$

$g_i(D(k))\sigma_{D(k)}U_{D(k)}^{1/p}| = O(\nu U_{i,j}^*)$. Combined with the prior paragraph, we have $U_{i,j}^* = O((|A_{i,1}| + |A_{i,2}|))$ as needed. Note that there are only $O(\log(n))$ terms i, j to union bound over, and from which the claim follows. ∎

Call the event where Claim 1 holds $\mathcal{E}_2$. Conditioned on $\mathcal{E}_2$, we can decompose $|A_{i,j}|/\mu_{i,j}$ for all i, j into $U_{i,j}^* + \mathcal{V}_{ij}$, where $\mathcal{V}_{ij}$ is some random variable satisfying $|\mathcal{V}_{ij}| = O(\nu(|A_{i,1}| + |A_{i,2}|))$ and $U_{i,j}^*$ is independent of the antirank vector D (it depends only on the hidden exponentials E_k and the uniformly random Gaussians $g_i(D(k))$). Now fix any realization of the count-max randomness, let $E = (E_1, \ldots, E_N)$ be the hidden exponential vector, $\mu = \{\mu_{i,1}, \mu_{i,2}\}_{i\in[d]}$, $D = (D(1), D(2), \ldots, D(N))$, and observe

$$\mathbf{Pr}\Big[\neg\texttt{FAIL} \mid D(1)\Big] = \sum_{E,\mu} \mathbf{Pr}\Big[\neg\texttt{FAIL} \mid D(1), E, \mu\Big]\mathbf{Pr}\Big[E, \mu\Big].$$

Here we have used the fact that E, μ are independent of the antiranks D. Thus, it will suffice to bound the probability of obtaining E, μ such that the event of failure can be determined by the realization of D. So consider any row i, and consider the event $\mathcal{Q}_i$ that $|\mu_{i,1}U_{i,1}^* - \mu_{i,2}U_{i,2}^*| < 2(|\mathcal{V}_{i,1}^*| + |\mathcal{V}_{i,2}^*|) = O(\nu(|A_{i,1}| + |A_{i,2}|)$ (where here we have conditioned on the high probability event $\mathcal{E}_2$). WLOG, $U_{i,1}^* \geq U_{i,2}^*$, giving $U_{i,1}^* = \Theta(|A_{i,1}| + |A_{i,2}|)$. Since the $\mu_{i,j}$'s are uniform, $\mathbf{Pr}[\mathcal{Q}_i] = O(\nu(|A_{i,1}| + |A_{i,2}|)/U_{i,1}^*) = O(\nu)$, and by a union bound $\mathbf{Pr}[\cup_{i\in[d]}\mathcal{Q}_i] = O(\log(n)\nu)$. Thus conditioned on $\mathcal{E}_1 \cap \mathcal{E}_2$ and $\neg(\cup_{i\in[d]}\mathcal{Q}_i)$, the event of failure is completely determined by the values E, μ and in particular is independent of the antirank vector D. Thus

$$\mathbf{Pr}\Big[\neg\texttt{FAIL} \,|D(1), E, \mu, \neg(\cup_{i\in[d]}\mathcal{Q}_i), \mathcal{E}_1 \cap \mathcal{E}_2\Big] = \mathbf{Pr}\Big[\neg\texttt{FAIL} \,|E, \mu, \neg(\cup_{i\in[d]}\mathcal{Q}_i), \mathcal{E}_1 \cap \mathcal{E}_2\Big].$$

So averaging over all E, μ,

$$\begin{aligned}\mathbf{Pr}\Big[\neg\texttt{FAIL} \,|D(1)\Big] &= \mathbf{Pr}\Big[\neg\texttt{FAIL} \,|D(1), \neg(\cup_{i\in[d]}\mathcal{Q}_i), \mathcal{E}_1 \cap \mathcal{E}_2\Big] + O(\log(n)\nu)\\ &= \mathbf{Pr}\Big[\neg\texttt{FAIL} \,|\neg(\cup_{i\in[d]}\mathcal{Q}_i), \mathcal{E}_1 \cap \mathcal{E}_2\Big] + O(\log(n)\nu)\\ &= \mathbf{Pr}\Big[\neg\texttt{FAIL}\Big] + O(\log(n)\nu)\end{aligned}$$

as needed. ∎

In Lemma 3, we demonstrated that the probability of failure can only change by an additive $\tilde{O}(\nu)$ term given that any one value of $i \in [N]$ achieved the maximum (i.e., $D(1) = i$). This property will translate into a $(1 \pm \tilde{O}(\nu))$-*relative error* in our sampler, where the space complexity is independent of ν. To complete the proof of correctness of our algorithm, we now need to bound the probability that we fail at all. To do so, we first prove the following fact about $\|z_{\text{tail(s)}}\|_2$, or the L_2 norm of z with the top s largest (in absolute value) elements removed.

PROPOSITION 1. *For any $s = 2^j \leq n^{c-2}$ for some $j \in \mathbb{N}$, we have $\sum_{i=4s}^N z_{D(i)}^2 = O(\|F\|_p^2/s^{2/p-1})$ if $p \in (0, 2)$ is a constant bounded below 2, and $\sum_{i=4s}^N z_{(i)}^2 = O(\log(n)\|F\|_p^2)$ if $p = 2$, with probability $1 - 3e^{-s}$.*

Proof. Let $I_k = \{i \in [N] \mid z_i \in (\frac{\|F\|_p}{2^{(k+1)/p}}, \frac{\|F\|_p}{2^{k/p}})\}$ for $k = 0, 1, \ldots, p\log(\|F\|_p)$ (where we have $\log(\|F\|_p^p) = O(\log(n))$). Note that $\mathbf{Pr}[i \in I_k] = \mathbf{Pr}[t_i \in (\frac{2^k F_i^p}{\|F\|_p^p}, \frac{2^{k+1}F_i^p}{\|F\|_p^p})]$ $< \frac{2^k F_i^p}{\|F\|_p^p}$, where the inequality follows from the fact that the pdf e^{-x} of the exponential distribution is upper bounded by 1, and the probability results from integrating the

pdf over an interval of size at most $\frac{2^k F_i^p}{\|F\|_p^p}$. Thus $\mathbb{E}[|I_k|] < 2^k$, so for every $k \geq \log(s) = j$, we have $\mathbf{Pr}[|I_k| > 4(2^k)] < e^{-s2^{k-j}}$ by Chernoff bounds. By a union bound, the probability that $|I_k| > 4(2^k)$ for any $k \geq \log(s)$ is at most $e^{-s}\sum_{i=0}^{O(\log(n))} e^{2^i} \leq 2e^{-s}$. Now observe $\mathbf{Pr}[z_i > \|F\|_p/s^{1/p}] < \frac{sF_i^p}{\|F\|_p^p}$, so $\mathbb{E}[|\{i|z_i > \|F\|_p/s^{1/p}\}|] < s$, and as before the number of such i with $z_i > \|F\|_p/s^{1/p}$ is at most $4s$ with probability $1-e^{-s}$. Conditioning on this, $\sum_{i=4s}^{N} z_{(i)}^2$ does not include the weight of any of these items, so

$$\sum_{i=4s}^{N} z_{(i)}^2 \leq \sum_{k=\log(s)}^{O(\log(n))} |I_k|\left(\frac{\|F\|_p}{2^{k/p}}\right)^2 \leq 4\sum_{k=0}^{O(\log(n))} \frac{\|F\|_p^2}{2^{(\log(s)+k)(2/p-1)}}.$$

First, if $p < 2$, the above sum is geometric and converges to at most $4\frac{\|F\|_p^2}{1-2^{-2/p+1}}\frac{1}{s^{2/p-1}} = O(\|F\|_p^2/s^{2/p-1})$ for p a constant bounded below by 2. If $p = 2$ or is arbitrarily close to 2, then each term is at most $\|F\|_p^2$, and the sum is upper bounded by $O(\log(n)\|F\|_p^2)$ as stated. Altogether, the probability of failure is at most $1-3e^{-s}$ by a union bound. ∎

LEMMA 4. *For $0 < p < 2$ a constant bounded away from 0 and 2, the probability that L_p `Sampler` outputs `FAIL` is at most $1-\Omega(1)$, and for $p = 2$ is $1-\Omega(1/\log(n))$.*

Proof. By Proposition 1, with probability $1 - 3e^{-4} > .9$ we have $\|z_{\text{tail}(16)}\|_2 = O(|F\|_p)$ for $p < 2$, and $\|z_{\text{tail}(16)}\| = O(\sqrt{\log(n)}\|F\|_p)$ when $p = 2$. Observe that for $t = 2, 3, \ldots, 16$, we have $|z_{D(t)}| < \|F\|_p(\frac{2}{\sum_{\tau=1}^{t} E_\tau})^{1/p}$, and with probability 99/100 we have $E_t > 1/100$, which implies that $|z_{D(t)}| = O(\|F\|_p)$ for all $t \in [16]$. Conditioned on this, we have $\|z_{\text{tail}(2)}\|_2 < q\|F\|_p$, where q is a constant when $p < 2$, and $q = \Theta(\sqrt{\log(n)})$ when $p = 2$. Now $|z_{D(1)}| = \frac{\|F\|_p}{E_1^{1/p}}$, and using the fact that the pdf exponential random variables around 0 are bounded above by a constant, we will have $|z_{D(1)}| > 20\|z_{-D(1)}\|_2$ with probability $\Omega(1)$ when $p < 2$, and probability $\Omega(\frac{1}{\log(n)})$ when $p = 2$. Conditioned on this, by Lemma 1, count-max will return the index $D(1)$ with probability $1 - n^{-c}$, and thus the sampling algorithm will not fail. ∎

Putting together the results of this section, we obtain the correctness of our algorithm as stated in Theorem 2. In section 5, we will show that the algorithm can be implemented to have $\tilde{O}(\nu^{-1})$ update and $\tilde{O}(1)$ query time and that the entire algorithm can be derandomized to use $O(\log^2 n(\log\log n)^2)$ bits of space for $p \in (0, 2)$ and $O(\log^3(n))$ bits for $p = 2$.

THEOREM 2. *Given any constant $c \geq 2$, $\nu \geq n^{-c}$, and $0 < p \leq 2$, there is a one-pass L_p sampler which returns an index $i \in [n]$ such that $\mathbf{Pr}[i = j] = \frac{|f_j|^p}{\|f\|_p^p}(1 \pm \nu) \pm n^{-c}$ for all $j \in [n]$, and which fails with probability $\delta > 0$. The space required is $O(\log^2(n)\log(1/\delta)(\log\log n)^2)$ bits for $p < 2$ and $O(\log^3(n)\log(1/\delta))$ bits for $p = 2$. For $p < 2$ and $\delta = 1/poly(n)$, the space is $O(\log^3(n))$-bits. The update time is $\tilde{O}(\nu^{-1})$, and the query time is $\tilde{O}(1)$.*

Proof. Conditioned on not failing, by Lemma 1, with probability $1-n^{-c}$ we have that the output $i_j \in [n^c]$ of count-max will in fact be equal to $\arg\max_i\{|\zeta_i|\}$. Recall that $\zeta_i = (1 \pm O(\nu))z_i$ for all $i \in [n^c]$ (and this rounding of z to ζ is order preserving). By Lemma 1 count-max only outputs a coordinate which is the *unique* maximizer of ζ. Now if there was a *unique* maximizer of ζ, there must also be a unique maximizer in z, from which it follows that $i_j = \arg\max_i\{|z_i|\}$.

Now Lemma 3 states for any $i_j \in [n^c]$ that $\mathbf{Pr}[\neg\texttt{FAIL} \mid i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}] = \mathbf{Pr}[\neg\texttt{FAIL}] \pm \tilde{O}(\nu) = q \pm \tilde{O}(\nu)$, where $q = \mathbf{Pr}[\neg\texttt{FAIL}] = \Omega(1)$ for $p < 2$, and $q =$

$\Omega(\frac{1}{\log(n)})$ for $p = 2$, both of which follow from Lemma 4, which does not depend on any of the randomness in the algorithm. Since conditioned on not failing, the output i_j of count-max satisfies $i_j = \arg\max_i\{|z_i|\}$, and the probability we output $i_j \in [n^c]$ is $\mathbf{Pr}[\neg\texttt{FAIL} \cap i_j = \arg\max\{|z_i|\}]$, so the probability our final algorithm outputs $i \in [n]$ is

$$\begin{aligned}\sum_{j\in[n^{c-1}]} \mathbf{Pr}[\neg\texttt{FAIL} \mid i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}]\mathbf{Pr}[i_j \\ = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}] = \sum_{j\in[n^{c-1}]} \frac{|f_i|^p}{\|F\|_p^p}(q \pm \tilde{O}(\nu)) \\ = \frac{|f_i|^p}{\|f\|_p^p}(q \pm \tilde{O}(\nu)).\end{aligned}$$

Note that we can scale the c value used in the algorithm by a factor of 60, so that the statement of Lemma 3 holds for any $\nu \geq n^{-c}$. The potential of the failure of the various high probability events that we conditioned on only adds another additive $O(n^{-c})$ term to the error. Thus, conditioned on an index i being returned, we have $\mathbf{Pr}[i = j] = \frac{|f_j|^p}{\|f\|_p^p}(1 \pm \tilde{O}(\nu))) \pm n^{-c}$ for all $j \in [n]$, which is the desired result after scaling ν by a poly$(\log(n))$ term. Running the algorithm $O(\log(\delta^{-1}))$ times in parallel for $p < 2$ and $O(\log(n)\log(\delta^{-1}))$ for $p = 2$, it follows that at least one index will be returned with probability $1 - \delta$.

For the complexity, the update time of count-max data structure A follows from the routine `Fast-Update` of Lemma 6, and the query time follows from Lemma 9. Theorem 7 shows that the entire algorithm can be derandomized to use a random seed with $O(\log^2(n)(\log\log(n))^2)$-bits, so to complete the claim it suffices to note that using $O(\log(n))$-bit precision as required by `Fast-Update` (Lemma 6), it follows that our whole data structure A can be stored with $O(\log^2(n))$ bits, which is dominated by the cost of storing the random seed. This gives the stated space after taking $O(\log(\delta^{-1}))$ parallel repetitions for $p < 2$. For $p = 2$, we only need a random seed of length $O(\log^3(n))$ for all $O(\log(n)\log(\delta^{-1}))$ repetitions by Corollary 4, which gives $O(\log^3(n)\log(\delta^{-1}) + \log^3(n)) = O(\log^3(n)\log(1/\delta))$ bits of space for $p = 2$ as stated. Similarly for the case of $p < 2$ and $\delta = 1/\text{poly}(n)$, the stated space follows from Corollary 4. ∎

In particular, it follows that perfect L_p samplers exist using $O(\log^2(n)\log(1/\delta)(\log\log n)^2)$ and $O(\log^3(n)\log(1/\delta))$ bits of space for $p < 2$ and $p = 2$, respectively.

THEOREM 3. *Given $0 < p \leq 2$, for any constant $c \geq 2$ there is a perfect L_p sampler which returns an index $i \in [n]$ such that $\mathbf{Pr}[i = j] = \frac{|f_j|^p}{\|F\|_p^p} \pm O(n^{-c})$ for all $j \in [n]$, and which fails with probability $\delta > 0$. The space required is $O(\log^2(n)\log(1/\delta)(\log\log n)^2)$ bits for $p < 2$, and $O(\log^3(n)\log(1/\delta))$ bits for $p = 2$. For $p < 2$ and $\delta = 1/poly(n)$, the space is $O(\log^3(n))$-bits.*

Finally, we note that the cause of having to pay an extra $(\log\log n)^2$ factor in the space complexity for $p < 2$ is only due to the derandomization. Thus, in the random oracle model where the algorithm has access to a poly(n)-length random tape which does not count against its space requirement, the space is an optimal $O(\log^2(n)\log(1/\delta))$. We remark that the $\Omega(\log^2(n)\log(1/\delta))$ of [KNP+17] lower bound also holds in the random oracle model.

COROLLARY 2. *For $p \in (0,2)$, in the random oracle model, there is a perfect L_p sampler which fails with probability $\delta > 0$ and uses $O(\log^2(n)\log(1/\delta))$ bits of space.*

Remark 1. Note that for p arbitrarily close to 2, the bound on $\|z\|_2$ of Proposition 1 as used in Lemma 4 degrades, as the sum of the L_2 norms of the level sets is no longer geometric and must be bounded by $O(\sqrt{\log(n)}\|F\|_2)$. In this case, the failure probability from Lemma 4 goes to $\Theta(\frac{1}{\log(n)})$, and so we must use the upper bound for $p = 2$. Similarly, for p arbitrarily close to 0, the bound also degrades since the values $V_{D(k)}$ in Lemma 3 blow up. For such nonconstant p arbitrarily close to 0, we direct the reader to the $O(\log^2(n))$-bit perfect L_0 sampler of [JST11].

5. Time and space complexity. In this section, we will show that our algorithm can be implemented with the desired space and time complexity. First, in section 5.1, we show how L_p `Sampler` can be implemented with the update procedure `Fast-Update` to result in $\tilde{O}(\nu^{-1})$ update time. Next, in section 5.2, we show that the algorithm L_p `Sampler` with `Fast-Update` can be derandomized to use a random seed of length $O(\log^2(n)(\log\log n)^2)$-bits, which will give the desired space complexity. Finally, in section 5.3, we show how using an additional heavy hitters data structure as a subroutine, we can obtain $\tilde{O}(1)$ update time as well. This additional data structure will not increase the space or update time complexity of the entire algorithm and does not need to be derandomized.

5.1. Optimizing the update time. In this section we prove Lemma 6. Our algorithm utilizes a single data structure run on the stream ζ, which is count-max matrix $A \in \mathbb{R}^{d\times 2}$, where $d = \Theta(\log(n))$. We will introduce an update procedure, `Fast-Update`, which updates the data structure A of L_p `Sampler` in $\tilde{O}(\nu^{-1})$ time. We assume the unit cost RAM model of computation, where a word of length $O(\log(n))$-bits can be operated on in $O(1)$ time. (Note that replacing $O(1)$ with $\mathrm{poly}(\log(n))$ time here would not affect our results, as the additional cost would be hidden in the $\tilde{O}$.) Throughout this section, we will refer to the *original algorithm* as the algorithm that implements L_p `sampler` by individually generating each scaling exponential t_i for $i \in [n^c]$ and hashing them individually into A (naïvely taking n^c update time). Our procedure will utilize the following result about efficiently sampling binomial random variables, which can be found in [BKP+14].

PROPOSITION 2. *For any constant $c > 0$, there is an algorithm that can draw a sample $X \sim \texttt{Bin}(n, 1/2)$ in expected $O(1)$ time in the unit cost RAM model. Moreover, it can be sampled in time $\tilde{O}(1)$ with probability $1 - n^{-c}$. The space required is $O(\log(n))$-bits.*

Proof. The proof of the running time bounds and correctness can be found in [BKP+14]. Since they do not analyze the space complexity of their routine, we do so here. Their algorithm is as follows. We can assume n is even; otherwise we could sample $\texttt{Bin}(n,q) \sim \texttt{Bin}(n-1,q) + \texttt{Bin}(1,q)$, where the latter can be sampled in constant time (unit cost RAM model) and $O(\log(n))$-bits of space. The algorithm first computes $\Delta \in [\sqrt{n}, \sqrt{n}+3]$, which can be done via any rough approximation of the function $\sqrt{x}$, and requires only $O(\log(n))$-bits. Define the block $\mathcal{B}_k = \{km, km+1, \ldots, km+m-1\}$ for $k \in \mathbb{Z}$, and set

$$f(i) = \frac{4}{2^{\max\{k,-k-1\}}m} \text{ s.t. } i \in \mathcal{B}_k,$$
$$p(i) = 2^{-n}\binom{n}{n/2+i}.$$

Note that given i, $f(i)$ can be computed in constant time and $O(\log(n))$ bits of space. The algorithm then performs the following loop:

1. Sample i via the normalized probability distribution $\bar{f} = f/16$.
2. Return $n/2 + i$ with probability $p(i)/f(i)$.
3. Else, reject i and return to step 1.

To compute the first step, the symmetry around $n/2$ of f is utilized. We flip unbiased coins $C_1, C_2, \ldots$ until we obtain C_{t+1} which lands tails, and pick i uniformly from block $\mathcal{B}_t$ or $\mathcal{B}_{-t}$ (where the choice is decided by a single coin flip). The procedure requires at most $O(\log(n))$-bits to store the index t. Next, to perform the second step, we obtain 2^{-L} additive error approximations $\tilde{q}$ of $q = (p(i)/f(i))$ for $L = 1, 2, \ldots$, which (using the fact that $0 \leq q \leq 1$) can be done by obtaining a 2^{-L}-relative error approximation of q. Then we flip L random bits to obtain a uniform $\tilde{R} \in [0, 1]$ and check if $|\tilde{R} - \tilde{q}| > 2^{-L}$. If so, we can either accept or reject i based on whether $\tilde{R} > \tilde{q} + 2^{-L}$ or not; otherwise we repeat with $L \leftarrow L + 1$.

To obtain $\tilde{q}$, it suffices to obtain a 2^{-L-1} relative error approximation of the factorial function $x!$. g, the 2^{-L} approximation

$$x! \approx (x+L)^{x+1/2} e^{-(x+L)} \left[\sqrt{2\pi} + \sum_{k=1}^{L-1} \frac{c_k}{x+k} \right],$$

is used, where $c_k = \frac{(-1)^{k-1}}{(k-1)!}(L-k)^{k-1/2} e^{L-k}$. This requires estimating the functions e^x, $\sqrt{x}$, and π, all of which, as well as each term in the sum, need only be estimated to $O(L)$-bits of accuracy (as demonstrated in [BKP+14]). Thus the entire procedure is completed in $O(L) = O(\log(n))$-bits of space (L can never exceed $O(\log(n))$, as q is specified with at most $O(\log(n))$ bits), which completes the proof. ∎

We now utilize a straightforward reduction from the case of sampling from $\texttt{Bin}(n, q)$ for any $q \in [0, 1]$ to sampling several times from $\texttt{Bin}(n', 1/2)$, where $n' \leq n$. This reduction has been observed before [FCT15], but we will state it here to clearly demonstrate our desired space and time bounds.

LEMMA 5. *For any constant $c > 0$ and $q \in [0, 1]$, there is an algorithm that can draw a sample $X \sim \texttt{Bin}(n, q)$ in expected $O(1)$ time in the unit cost RAM model. Moreover, it can be sampled in time $\tilde{O}(1)$ with probability $1 - n^{-c}$, and the space required is $O(\log(n))$-bits.*

Proof. The reduction is as follows (for a more detailed proof of correctness, see [FCT15]). We sample $\texttt{Bin}(n, q)$ by determining how many of the n trials were successful. This can be done by generating variables $u_1, \ldots, u_n$ uniform on $[0, 1]$ and determining how many are less than q. We do this without generating all the variables u_i explicitly as follows. First write q in binary as $q = (0.q_1 q_2, \ldots)_2$. Set $b \leftarrow 0$, $j \leftarrow 1$, $n_j \leftarrow n$ and sample $b_j \sim \texttt{Bin}(n_j, 1/2)$. If $q_j = 1$, then set $b = b + b_j$, as these corresponding b_j trials u_i with the first bit set to 0 will all be successful trials given that $q_j = 1$. Then set $n_{j+1} \leftarrow n_j - b_j$ and repeat with $j \leftarrow j + 1$. Otherwise, if $q_j = 0$, then we set $n_{j+1} \leftarrow n_j - (n_j - b_j) = b_j$, since this represents the fact that $(n_j - b_j)$ of the variables u_i will be larger than q. With probability $1 - n^{-100c}$, we reach the point where $n_j = 0$ within $O(\log(n))$ iterations, and we return the value stored in b at this point. By Proposition 2, each iteration requires $\tilde{O}(1)$ time, and thus the entire procedure is $\tilde{O}(1)$. For space, note that we need only store q to its first $O(\log(n))$ bits, since the procedure terminates with high probability within $O(\log(n))$ iterations. Then the entire procedure requires $O(\log(n))$ bits, since each sample of $\texttt{Bin}(n_j, 1/2)$ requires only $O(\log(n))$ space by Proposition 2. ∎

The `Fast-Update` *procedure.* We are now ready to describe the implementation of our algorithm's update procedure. Specifically, our goal is to show that the update time of our algorithm can be made $\tilde{O}(1/\nu)$, where ν is the relative error in the sampler. Now recall that for our *perfect* L_p sampler we require $\nu = 1/\text{poly}(n)$, and thus the `Fast-Update` procedure will not improve the update time of our perfect L_p sampler (beyond the naïve poly(n)). However, if one allows the relative error ν to be larger (i.e., an approximate sampler), then a $\tilde{O}(1/\nu)$ update time is now much faster. Thus, the `Fast-Update` procedure allows for a trade-off between the update time and the relative error of the sampler. Note that all prior works had a dependency on the relative error ν in *both* the spacial complexity and the update time of the sampler.[3]

Recall that our algorithm utilizes just a single data structure on the stream ζ: the $d \times 2$ count-max matrix A (where $d = \Theta(\log(n))$). Upon receiving an update (i, Δ) to a coordinate f_i for $i \in [n]$, we proceed as follows. Our goal is to compute the set $\{\texttt{rnd}_\nu(1/t_{i_1}^{1/p}), \texttt{rnd}_\nu(1/t_{i_2}^{1/p}), \ldots, \texttt{rnd}_\nu(1/t_{i_{n^{c-1}}}^{1/p})\}$ and update each row of A accordingly in $\tilde{O}(\nu^{-1})$ time, where ν is the error parameter for L_p sampling which will factor only into the update time of the algorithm and not the space complexity. Naïvely, this could be done by computing each value individually and then updating each row of A accordingly, but this would require $O(n^{c-1})$ time. To avoid this and obtain speed-ups when the relative error ν is made larger than $1/\text{poly}(n)$, we exploit the fact that the support size of $\texttt{rnd}_\nu(x)$ for $1/\text{poly}(n) \le x \le \text{poly}(n)$ is $\tilde{O}(\nu^{-1})$, so it will suffice to determine how many variables $\texttt{rnd}_\nu(1/t_{i_j}^{1/p})$ are equal to each value in the support of $\texttt{rnd}_\nu(x)$.

Our update procedure is then as follows. Let $I_j = (1+\nu)^j$ for $j = -\Pi, -\Pi+1, \ldots, \Pi-1, \Pi$, where $\Pi = O(\log(n)\nu^{-1})$. We utilize the c.d.f. $\psi(x) = 1 - e^{-x^{-p}}$ of the $1/p$th power of the inverse exponential distribution $t^{-1/p}$ (here t is exponentially distributed). Then beginning with $j = -\Pi, -\Pi+1, \ldots, \Pi$ we compute the probability $q_j = \psi(I_{j+1}) - \psi(I_j)$ that $\texttt{rnd}_\nu(1/t^{1/p}) = I_j$ and then compute the number of values Q_j in $\{\texttt{rnd}_\nu(1/t_{i_1}^{1/p}), \texttt{rnd}_\nu(1/t_{i_2}^{1/p}), \ldots, \texttt{rnd}_\nu(1/t_{i_{n^{c-1}}}^{1/p})\}$ which are equal to I_j. With probability $1 - n^{100c}$, we know that $1/\text{poly}(n) \le t_i \le \text{poly}(n)$ for all $i \in [N]$, and thus conditioned on this, we will have completely determined the values of the items in $\{\texttt{rnd}_\nu(1/t_{i_1}^{1/p}), \texttt{rnd}_\nu(1/t_{i_2}^{1/p}), \ldots, \texttt{rnd}_\nu(1/t_{i_{n^{c-1}}}^{1/p})\}$ by looking at the number equal to I_j for $j = -\Pi, \ldots, \Pi$.

Now we know that there are Q_j updates which we need to hash into A (along with i.i.d. Gaussian scalings), each with the same value ΔI_j. This is done by the procedure `Fast-Update-CS` (Figure 4), which computes the number of updates $b_{k,\theta}$ that hashes to each bucket $A_{k,\theta}$ by drawing binomial random variables. Once this is done, we know that the value of $A_{k,\theta}$ should be updated by the value $\sum_{t=1}^{b_{k,\theta}} g_t \Delta I_j$, where each $g_t \sim \mathcal{N}(0,1)$. Naïvely, computing the value $\sum_{t=1}^{b_{k,\theta}} g_t \Delta I_j$ would involve generating $b_{k,\theta}$ random Gaussians. To avoid this, we utilize the 2-stability of Gaussians (Definition 2), which asserts that $\sum_{t=1}^{b_{k,\theta}} g_t \Delta I_j \sim g\sqrt{b_{k,\theta}}\Delta I_j$, where $g \sim \mathcal{N}(0,1)$. Thus we can simply generate and store the Gaussian g associated with the item $i \in [n]$, rounding I_j, and bucket $A_{k,\theta}$, and on each update Δ to f_i we can update $A_{k,\theta}$ by $g\sqrt{b_{k,\theta}}\Delta I_j$.

Finally, once the number of values in

$$\{\texttt{rnd}_\nu(1/t_{i_1}^{1/p}), \texttt{rnd}_\nu(1/t_{i_2}^{1/p}), \ldots, \texttt{rnd}_\nu(1/t_{i_{n^{c-1}}}^{1/p})\}$$

[3]Note that our algorithm has no dependency on ν in the spacial complexity as long as $1/\nu = O(\text{poly}(n))$.

`Fast-Update-CS` (A, Q, I, Δ, i)
Set $W_k = Q$ for $k = 1, \ldots, d$
`For` $k = 1, \ldots, d$,

1. `For` $\theta = 1, 2$:
   (a) Draw $b_{k,\theta} \sim \texttt{Bin}(W_k, \frac{1}{2-\theta+1})$.
   (b) Draw and store $g_{k,\theta,I,i} \sim \mathcal{N}(0,1)$. Reuse on every call to `Fast-Update-Cs` with the same parameters (k, θ, I, i).
   (c) Set $A_{k,\theta} \leftarrow A_{k,\theta} + g_{k,\theta,I,i}\sqrt{b_{k,\theta}}\Delta I$.
   (d) $W_k \leftarrow W_k - b_{k,\theta}$.

FIG. 4. *Update A via updates to Q coordinates, each with a value of ΔI.*

`Fast-Update` (i, Δ, A)
Set $L = n^{c-1}$, and fix $K = \Theta(\log(n))$ with a large enough constant.
`For` $j = -\Pi, -\Pi + 1, \ldots, \Pi - 1, \Pi$:

1. Compute $q_j = \psi(I_{j+1}) - \psi(I_j)$.
2. Draw $Q_j \sim \texttt{Bin}(L, q_j)$.
3. **If** $L < K$, hash the Q_j items individually into each row A_ℓ using explicitly stored uniform i.i.d. random variables $h_\ell : [n^c] \to [2]$ and Gaussians $g_\ell(j)$ for $\ell \in [d]$.
4. **Else:** update count-max table A by via `Fast-Update-CS`(A, Q_j, I_j, Δ, i).
5. $L \leftarrow L - Q_j$.

FIG. 5. *Algorithm to Update count-max A.*

that are left to determine is less than K for some $K = \Theta(\log(n))$, we simply generate and hash each of the remaining variables individually. The generation process is the same as before, except that for each of these at most K remaining items we associate a fixed index i_j for $j \in [n^{c-1}]$ and store the relevant random variables $h_\ell(i_j), g_\ell(i_j)$ for $\ell \in [d]$. Since the value of j that is chosen for each of these coordinates does not affect the behavior of the algorithm—in other words the index of the duplicate that is among the K largest is irrelevant—we can simply choose these indices to be $i_1, i_2, \ldots, i_K \in [N]$ so that the first item hashed individually via step 3 corresponds to ζ_{i_1}, the second to ζ_{i_2}, and so on.

Note that the randomness used to process an update corresponding to a fixed $i \in [n]$ is stored so it can be reused to generate the same updates to A whenever an update to i is made. Thus, each time an update $+1$ is made to a coordinate $i \in [n]$, each bucket of count-max is updated by the same value. When an update of size Δ comes, this update to the count-max buckets is scaled by Δ. For each $i \in [n]$, let K_i denote the size of L when step 3 of Figure 5 was first executed while processing an update to i. In other words, the coordinates $\zeta_{i_1}, \ldots, \zeta_{i_{K_i}}$ were hashed into each row $\ell \in [d]$ of A using explicitly stored random variables $h_\ell(i_j), g_\ell(i_j)$. Let $\mathcal{K} = \cup_{i \in [n]} \cup_{j=1}^{K_i} \{i_j\}$. Then on the termination of the algorithm, to find the maximizer of ζ, the count-max algorithm checks for each $i \in \mathcal{K}$, whether i hashed to the largest bucket (in absolute value) in a row at least a $\frac{4}{5}$ fraction of the time. Count-max then returns the first i that satisfies this, or `FAIL`. In other words, the count-max algorithm decides to fail or output an index i based on computing the fraction of rows for which i hashes into the largest bucket, except now it only computes these values for $i \in \mathcal{K}$ instead of $i \in [n^c]$; thus count-max can only return a value of $i \in \mathcal{K}$. We now argue

that the distribution of our algorithm is not changed by using the update procedure `Fast-Update`. This will involve showing that $\arg\max\{|\zeta_i|\} \in \mathcal{K}$ if our algorithm was to return a coordinate originally.

LEMMA 6. *Running the L_p `sampler` with the update procedure given by `Fast-Update` results in the same distribution over the count-max table A as the original algorithm. Moreover, conditioned on a fixed realization of A, the output of the original algorithm will be the same as the output of the algorithm using `Fast-Update`. For a given $i \in [n]$, `Fast-Update` requires $\tilde{O}(\nu^{-1})$-random bits and runs in time $\tilde{O}(\nu^{-1})$.*

Proof. To hash an update Δ to a coordinate f_i, the procedure `Fast-Update` computes the number Q_j of variables in the set $\{\mathtt{rnd}_\nu(1/t_{i_1}^{1/p}), \mathtt{rnd}_\nu(1/t_{i_2}^{1/p}), 2\ldots, \mathtt{rnd}_\nu(1/t_{i_{n^{c-1}}}^{1/p})\}$ which are equal to I_j for each $j \in \{-\Pi, \ldots, \Pi\}$. Instead of computing Q_j by individually generating the variables and rounding them, we utilize a binomial random variable to determine Q_j, which results in the same distribution over $\{\mathtt{rnd}_\nu(1/t_{i_1}^{1/p}), \mathtt{rnd}_\nu(1/t_{i_2}^{1/p}), 2\ldots, \mathtt{rnd}_\nu(1/t_{i_{n^{c-1}}}^{1/p})\}$. As noted, with probability $1 - n^{100c}$ none of the variables $\mathtt{rnd}_\nu(1/t_{i_j}^{1/p})$ will be equal to I_k for $|k| > \Pi$, which follows from the fact that $n^{-101c} < t_i < O(\log(n))$ with probability $1 - n^{-101c}$ and then union bounding over all n^c exponential variables t_i. So we can safely ignore this low probability event.

Once computed, we can easily sample from the number of items of the Q_j that go into each bucket $A_{k,\theta}$, which is the value $b_{k,\theta}$ in `Fast-Update-CS` (Figure 4). By 2-stability of Gaussians (Definition 2), we can update each bucket $A_{k,\theta}$ by $g_{k,\theta,I_j,i}\sqrt{b_{k,\theta}}\Delta I_j$, which is distributed precisely the same as if we had individually generated each of the $b_{k,\theta}$ Gaussians and taken their inner product with the vector $\Delta I_j \vec{1}$, where $\vec{1}$ is the all 1's vector. Storing the explicit values $h_\ell(i_j)$ for the top K largest values of $\mathtt{rnd}_\nu(1/t_{i_j}^{1/p})$ does not affect the distribution but only allows the algorithm to determine the induces of the largest coordinates i_j corresponding to each $i \in [n]$ at the termination of the algorithm. Thus the distribution of updates to A is unchanged by the `Fast-Update` procedure.

We now show that the output of the algorithm run with this update procedure is the same as it would have been had all the random variables been generated and hashed individually. First observe that for $\nu < 1/2$, no value $q_j = \psi(I_{j+1}) - \psi(I_j)$ is greater than $1/2$. Thus at any iteration, if $L > K$, then $L - \mathtt{Bin}(L, q_j) > L/3$ with probability $1 - n^{-100c}$ by Chernoff bounds (using that $K = \Omega(\log(n))$). Thus for the first iteration at which L drops below K, we will have $L > K/3$. So for each $i \in [n]$ the top $K/3$ values ζ_{i_j} will be hashed into each row A_ℓ using stored random variables $h_\ell(i_j)$, so $K_i > K/3 = \Omega(\log(n))$ for all $i \in [n]$. In particular, $K_i > 0$ for all $i \in [n]$.

Now the only difference between the output procedure of the original algorithm and that of the efficient-update time algorithm is that in the latter we only compute the values of $\alpha_{i_j} = \left|\{t \in [d] \mid |A_{t,h_t(i_j)}| = \max_{r\in\{1,2\}} |A_{t,r}|\}\right|$ for the $i_j \in [n^c]$ corresponding to the K_i largest values $t_{i_j}^{-1/p}$ in the set $\{t_{i_1}^{-1/p}, \ldots, t_{i_{n^{c-1}}}^{-1/p}\}$, whereas in the former all values of α_{i_j} are computed to find a potential maximizer. In other words, count-max with `Fast-Update` only searches through the subset $\mathcal{K} \subset [n^c]$ for a maxmizer instead of searching through all of $[n^c]$ (here $\mathcal{K}$ is as defined earlier in this section). Since count-max never outputs a index i_j that is not a unique maximizer with high probability, we know that the output of the original algorithm, if it does not fail, must be i_j such that $j = \arg\max_{j'}\{t_{i_{j'}}\}$, and therefore $i_j \in \mathcal{K}$. Note the n^{-c}

failure probability can be safely absorbed into the additive n^{-c} error of the perfect L_p sampler. Thus the new algorithm will also output i_j. Since the new algorithm with `Fast-Update` searches over the subset $\mathcal{K} \subset [n^c]$ for a maximier, if the original algorithm fails, then certainly so will `Fast-Update`. Thus the output of the algorithm using `Fast-Update` is distributed identically (up to n^{-c} additive error) as the output of the original algorithm, which completes the proof. ∎

Runtime & Random Bits. For the last claim, first note that it suffices to generate all continuous random varaibles used up to $(nmM)^{-c} = 1/\text{poly}(n)$ precision, which is $1/\text{poly}(n)$ additive error after conditioning on the event that all random variables are all at most $\text{poly}(n)$ (which occurs with probability $1 - n^{-c}$) and recalling that the length of the stream m satisfies $m < \text{poly}(n)$ for a suitably smaller $\text{poly}(n)$ then as in the additive error. More formally, we truncate the binary representation of every continuous random variable (both the exponentials and Gaussians) after $O(\log(n))$-bits with a sufficiently large constant. This will result in at most an additive $1/\text{poly}(n)$ error for each bucket $A_{i,j}$ of A, which can be absorbed by the adversarial error $\mathcal{V}_{i,j}$ with $|\mathcal{V}_{i,j}| = O(\nu(|A_{i,1}| + |A_{i,2}|))$ that we incur in each of these buckets already in Lemma 3. Thus each random variable requires $O(\log(n))$ bits to specify. Similarly, a precision of at most $(nmM)^{-c}$ is needed in the computation of the q_j's in Figure 5 by Lemma 5, since the routine to compute $\texttt{Bin}(n, q_j)$ will terminate with probability $1 - n^{-100c}$ after querying at most $O(\log(n))$ bits of q_j. Now there are at most $2\Pi = O(\nu^{-1}\log(n))$ iterations of the loop in `Fast-Update`. Within each, our call to sample a binomial random variable is carried out in $\tilde{O}(1)$ time with high probability by Lemma 5 (and thus use at most $\tilde{O}(1)$ random bits), and there are $\tilde{O}(1)$ entries in A to update (which upper bounds the running time and randomness requirements of `Fast-Update-CS`).

Note that since the stream has length $m = \text{poly}(n)$, and there are at most $\tilde{O}(\nu)$ calls made to sample binomial random variables in each, we can union bound over each call to guarantee that each returns in $\tilde{O}(1)$ time with probability $1 - n^{-100c}$. Since $K = \tilde{O}(1)$, we must store an additional $\tilde{O}(1)$ random bits to store the individual random variables $h_\ell(i_j)$ for $i_j \in \{i_1, \ldots, i_{K_i}\}$. Similarly, we must store $\tilde{O}(\nu)$ independent Gaussians for the procedure `Fast-Update-CS`, which also terminates in $\tilde{O}(1)$ time, which completes the proof. ∎

5.2. Derandomizing the algorithm. We now show that our algorithm L_p `Sampler` with `Fast-Update` can be derandomized without affecting the space or time complexity. Recall that our main L_p sampling algorithm utilizes two main sources of randomness. First, it uses randomness to generate the exponential random scaling variables $(t_1, \ldots, t_{n^c})$ (the "exponential randomness"), and second, it uses randomness to generate the Gaussian coefficients $g_i(j)$ and fully random hash functions $h_i(j)$ needed for count-max (the "count-max randomness"). To derandomize both these sources of randomness, we will need to use a combination of Nisan's PRG [Nis92] and the PRG of Goplan, Kane, and Meka [GKM18]. Specifically, we will derandomize the exponential randomness with the PRG of Goplan, Kane, and Meka, and we will derandomize the count-max randomness with Nisan's PRG.

We begin by introducing Nisan's PRG, which is a deterministic map $G : \{0,1\}^\ell \to \{0,1\}^T$, where $T \gg \ell$ (for instance, one can think of $T = \text{poly}(n)$ and $\ell = O(\log^2(n))$). Let $\sigma : \{0,1\}^T \to \{0,1\}$ be a tester (a function computable under some specified restrictions). For the case of Nisan's PRG, σ must be a tester which reads its random T-bit input in a stream, left to right, and outputs either 0 or 1 at the end. Nisan's

PRG can be used to fool *any* such tester, which means

$$\left|\mathbf{Pr}[\sigma(U_T) = 1] - \mathbf{Pr}[\sigma(G(U_\ell)) = 1]\right| < \frac{1}{T^c},$$

where U_t indicates t uniformly random bits for any t, and c is a sufficiently large constant. Here the probability is taken over the choice of the random bits U_T and U_ℓ. In other words, the probability that σ outputs 1 is nearly the same when it is given random input as opposed to input from Nisan's generator. Note that since $\sigma(U_T)$ is a $0,1$ random variable, the left-hand side of the above equation can be rewritten as $|\mathbb{E}[\sigma(U_T)] - \mathbb{E}[\sigma(G(U_\ell))]|$. Nisan's theorem states if σ can be computed by an algorithm with at most poly(T) states and which uses a working memory tape of size at most $O(\log T)$, then a seed length of $\ell = O(\log^2 T)$ suffices for the above result [Nis92]. More generally, if σ can be computed by an algorithm with poly(T) states and which uses a working memory of tape of size $S = \Omega(\log T)$, then a seed length of $\ell = O(S \log T)$ suffices for the above result. Thus Nisan's PRG fools space bounded testers σ that read their randomness in a stream.

Why Nisan's PRG alone is insufficient. We remark that it is possible to derandomize our entire algorithm with Nisan's PRG, albeit with suboptimal seed length. Since our algorithm is a linear sketch and is therefore independent of the ordering of the stream, one can assume for the sake of the derandomization that the stream is ordered so that all updates to a single coordinate occur consecutively (this is a fairly standard argument, e.g., [Ind06]). Reading the exponential and count-max randomness in a stream, one can then fully construct the state of the algorithm's data structure at the end of the stream, by adding the contribution of each coordinate to the whole data structure one by one. The space required to do this is the size of the data structure, which is $O(\log^2 n)$ bits. Then to derandomize with Nisan's PRG, we would require a seed length of $O(\log^3 n)$-bits, which does not match our desired space complexity. Thus to improve the seed length to $O(\log^2 n(\log\log n)^2)$, we will need the approach followed here.

We remark that the main difficulty in applying Nisan's PRG alone is that, for a given i, to test if i is returned by count-max, one must check for each pair of buckets in count-max whether i hashes to the larger bucket. Since each bucket depends on the same exponential randomness, one would either need to make multiple passes over the exponential randomness (which is not allowed by Nisan's), once for each bucket, or one would need to store all the buckets simultaneously. On the other hand, if the exponential randomness was *fixed*, and hard-coded into the tester σ, then one could construct the buckets one at a time, reading only the count-max randomness in a stream, and thus only using $O(\log n)$ bits of space. We make use of this latter fact, by generating the exponential randomness with a separate PRG from [GKM18], to obtain our main result.

5.2.1. Road-map for the derandomization. We now briefly lay out the structure of this section. First, in section 5.2.2, we introduce the PRG of Goplan, Kane, and Meka [GKM18], along with the notion of a half-space tester, which will be crucial for us. We then demonstrate in Lemma 7 how the PRG of [GKM18] can be used to fool such half-space testers with a small seed length. Recall the PRG is just a function $G : \{0,1\}^\ell \to \{-M, \ldots, M\}^n$ for some $M = \text{poly}(n)$. However, in order to use this PRG for our streaming algorithm, we must show not only that the seed length is small, but also that given a seed x, one can compute each coordinate of $G(x)$ in small space *and* small runtime, which we do in Proposition 3. We then use this fact

that G can fool half-space testers to prove a set of general results about derandomizing streaming algorithms, which is captured in Theorem 5. As a corollary, we obtain a novel and improved derandomization of the count-sketch variant of [MP14]. Finally, in section 5.2.4, we present our main theorem, Theorem 7, which uses a combination of the PRG of Goplan, Kane, and Meka as well as Nisan's PRG to derandomize our streaming algorithm.

5.2.2. Half space fooling PRG's. Our derandomization crucially uses the PRG of Goplan, Kane, and Meka [GKM18], which fools a certain class of Fourier transforms. Utilizing the results of [GKM18], we will design a PRG that can fool *arbitrary* functions of $\lambda = O(\log(n))$ halfspaces, using a seed of length $O(\log^2(n)(\log\log(n))^2)$. We remark that in [GKM18] it is shown how to obtain such a PRG for a function of a single half-space. Using extensions of the techniques in that paper, we demonstrate that the same PRG with a smaller precision ϵ can be used to fool functions of more half-spaces. We now introduce the main result of [GKM18]. Let $\mathbb{C}_1 = \{c \in \mathbb{C} \mid |c| \leq 1\}$.

DEFINITION 4 (Definition 1 [GKM18]). *An (m,n)-Fourier shape $f : [m]^n \to \mathbb{C}_1$ is a function of the form $f(x_1,\ldots,x_n) = \prod_{j=1}^n f_j(x_j)$, where each $f_j : [m] \to \mathbb{C}_1$.*

THEOREM 4 (Theorem 1.1 [GKM18]). *There is a PRG $G : \{0,1\}^\ell \to [m]^n$ that fools all (m,n)-Fourier shapes f with error ϵ using a seed of length*

$$\ell = O(\log(mn/\epsilon)(\log\log(mn/\epsilon))^2),$$

meaning

$$\Big|\mathbb{E}\Big[f(x)\Big] - \mathbb{E}\Big[f(G(y))\Big]\Big| \leq \epsilon,$$

where x is uniformly chosen from $[m]^n$ and y from $\{0,1\}^\ell$.

For any $a^1,\ldots,a^\lambda \in \mathbb{Z}^n$ and $\theta_1,\ldots,\theta_\lambda \in \mathbb{Z}$, let $H_i : \mathbb{R}^n \to \{0,1\}$ be the function given by $H_i(X_1,\ldots,X_n) = \mathbf{1}[a_1^i X_1 + a_2^i X_2 + \cdots + a_n^i X_n > \theta_i]$, p where $\mathbf{1}$ is the indicator function. We now define the notion of a λ-halfspace tester and what it means to fool one.

DEFINITION 5 (λ-halfspace tester). *A λ-halfspace tester is any function $\sigma_H : \mathbb{R}^n \to \{0,1\}$ which, on input $X = (X_1,\ldots,X_n)$, outputs $\sigma'_H(H_1(X),\ldots,H_\lambda(X)) \in \{0,1\}$, where σ'_H is any fixed function $\sigma'_H : \{0,1\}^\lambda \to \{0,1\}$. In other words, the Boolean valued function $\sigma_H(X)$ only depends on the values $(H_1(X),\ldots,H_\lambda(X))$. A λ-halfspace tester is said to be M bounded if all the half-space coefficents a_j^i and θ_i are integers of magnitude at most M, and each X_i is drawn from a discrete distrubtion $\mathcal{D}$ with support contained in $\{-M,\ldots,M\} \subset \mathbb{Z}$.*

DEFINITION 6 (fooling a λ-halfspace tester). *A PRG $G : \{0,1\}^\ell \to \mathbb{R}^n$ is said to ϵ-fools the class of λ-halfspace testers under a distribution $\mathcal{D}$ over $\mathbb{R}^n$ if for every set of λ halfspaces $H = (H_1,\ldots,H_\lambda)$ and every λ-halfspace tester $\sigma_H : \mathbb{R}^n \to \{0,1\}$, we have*

$$\big|\mathbb{E}_{X\sim\mathcal{D}}\big[\sigma_H(X) = 1\big] - \mathbb{E}_{y\sim\{0,1\}^\ell}\big[\sigma_H(G(y)) = 1\big]\big| < \epsilon.$$

Here ℓ is the seed length of G.

We will consider only product distributions $\mathcal{D}$. In other words, we assume that each coordiante X_i is drawn i.i.d. from a fixed distribution $\mathcal{D}$ over $\{-M,\ldots,M\} \subset \mathbb{Z}$. We consider PRG's $G : \{0,1\}^\ell \to \{-M,\ldots,M\}^n$ which take in a random seed of length ℓ and output a $X' \in \{-M,\ldots,M\}^n$ such that any M-bounded λ-halfspace

tester will be unable to distinguish X' from $X \sim \mathcal{D}^n$ (where $\mathcal{D}^n$ is the product distribution of $\mathcal{D}$, such that each $X_i \sim \mathcal{D}$ independently). The following lemma demonstrates that the PRG of [GKM18] can be used to fool M-bounded λ-halfspace testers. The authors would like to thank Raghu Meka for providing us with a proof of Lemma 7.

LEMMA 7. *Suppose $X_i \sim \mathcal{D}$ is a distribution on $\{-M, \dots, M\}$ that can be sampled from with $\log(M') = O(\log(M))$ random bits. Then, for any $\epsilon > 0$ and constant $c \geq 1$, there is a PRG $G : \{0,1\}^\ell \to \{-M, \dots, M\}^n$ which $\epsilon(nM)^{-c\lambda}$-fools the class of all M-bounded λ-halfspace testers on input $X \sim \mathcal{D}^n$ with a seed of length $\ell = O(\lambda \log(nM/\epsilon)(\log\log(nM/\epsilon))^2)$ (assuming $\lambda \leq n$). Moreover, if $G(y) = X' \in \{-M, \dots, M\}^n$ is the output of G on random seed $y \in \{0,1\}^\ell$, then each coordinate X'_i can be computed in $O(\ell)$-space and in $\tilde{O}(1)$ time, where $\tilde{O}$ hides poly$(\log(nM))$ factors.*

Proof. Let $X = (X_1, \dots, X_n)$ be uniformly chosen from $[M']^n$ for some $M' = \text{poly}(M)$, and let $Q : [M'] \to \{-M, \dots, M\}$ be such that $Q(X_i) \sim \mathcal{D}^n$ for each $i \in [n]$. Let $a^1, \dots, a^\lambda \in \mathbb{Z}^n$, $\theta_1, \dots, \theta_\lambda \in \mathbb{Z}$ be $\log(M)$-bit integers, where $H_i(x) = \mathbf{1}[\langle a^i, x\rangle > \theta_i]$. Let $Y_i = \langle Q(X), a^i\rangle - \theta_i$. Note that $Y_i \in [-2M^2n, 2M^2n]$. So fix any $\alpha_i \in [-2M^2n, 2M^2n]$ for each $i \in [\lambda]$, and let $\alpha = (\alpha_1, \dots, \alpha_\lambda)$. Let $h_\alpha(x) = \mathbf{1}(Y_1 = \alpha_1) \cdot \mathbf{1}(Y_2 = \alpha_2) \cdots \mathbf{1}(Y_\lambda = \alpha_\lambda)$, where $\mathbf{1}(\cdot)$ is the indicator function. Now define $f(x) = \sum_{j=1}^{\lambda} (2M^2n)^{j-1} \langle a^i, x\rangle$ for any $x \in \mathbb{Z}^n$. Note that $f(Q(X)) \in \{-(Mn)^{O(\lambda)}, \dots, (Mn)^{O(\lambda)}\}$. We define the Kolmogorov distance between two integer valued random variables Z, Z' by $d_K(Z, Z') = \max_{k \in \mathbb{Z}}(|\mathbf{Pr}[Z \leq k] - \mathbf{Pr}[Z' \leq k]|)$. Let $X' \in [M']^n$ be generated via the (M', n)-Fourier shape PRG of [GKM18] with error ϵ' (Theorem 1.1 [GKM18]). Observe $\mathbb{E}[h_\alpha(Q(X))] = \mathbf{Pr}[f(Q(X)) = \sum_{j=1}^{\lambda} (Mn)^{j-1}\alpha_j]$, so

$$|\mathbb{E}[h_\alpha(Q(X))] - \mathbb{E}[h_\alpha(Q(X')]| \leq d_K(f(Q(X)), f(Q(X'))).$$

Now by Lemma 9.2 of [GKM18],

$$d_K(f(Q(X)), f(Q(X'))) = O\big(\lambda \log(Mn) d_{FT}\big(f(Q(X)), f(Q(X'))\big)\big),$$

where for integer valued Z, Z', we define

$$d_{FT}(Z, Z') = \max_{\beta \in [0,1]} |\mathbb{E}[\exp(2\pi i \beta Z)] - \mathbb{E}[\exp(2\pi i \beta Z')]|.$$

Now $\exp(2\pi i \beta f(Q(X))) = \prod_{i=1}^{n}((\sum_{j=1}^{\lambda}(2M^2n)^{j-1}a_i^j)Q(X_i))$, which is a (M', n)-Fourier shape as in Definition 4. Thus by Theorem 4 (Theorem 1.1 of [GKM18]), we have $d_{FT}(f(Q(X)), f(Q(X'))) \leq \epsilon'$. Thus

$$|\mathbb{E}[h_\alpha(Q(X))] - \mathbb{E}[h_\alpha(Q(X')]| = O(\lambda \log(Mn)\epsilon').$$

Now let $\sigma_H(x) = \sigma'_H(H_1(x), \dots, H_\lambda(x))$ be any M-bounded λ-halfspace tester on $x \sim \mathcal{D}^n$. Since the inputs to the halfspaces H_i of σ'_H are all integers in $\{-2M^2n, 2M^2n\}$, let $A \subset \{-2M^2n, 2M^2n\}$ be the set of $\alpha \in A$ such that $Y = (Y_1, \dots, Y_\lambda) = \alpha$ implies that $\sigma_H(Q(X)) = 1$, where $Q(X) \sim \mathcal{D}^n$ as above. Recall here that $Y_i = \langle Q(X), a^i\rangle - \theta_i$. Then we can think of a $\sigma_H(X) = \sigma''_H(Y_1, \dots, Y_\lambda)$ for some func-

tion $\sigma''_H : \{-2M^2n, \ldots, 2M^2n\}^\lambda \to \{0,1\}$, and in this case we have $A = \{\alpha \in \{-2M^2n, 2M^2n\} \mid \sigma''_H(\alpha) = 1\}$. Then

$$|\mathbb{E}[\sigma_H(Q(X))] - \mathbb{E}[\sigma_H(Q(X'))]| \leq \sum_{\alpha \in A} |\mathbb{E}[h_\alpha(Q(X))] - \mathbb{E}[h_\alpha(Q(X')]|$$

$$\leq \sum_{\alpha \in A} O(\lambda \log(Mn)\epsilon').$$

Now note that $|A| = (nM)^{O(\lambda)}$, so setting $\epsilon' = \epsilon(nM)^{-O(\lambda)}$ with a suitably large constant, we obtain $|\mathbb{E}[\sigma_H(Q(X))] - \mathbb{E}[\sigma_H(Q(X'))]| \leq \epsilon(nM)^{-c\lambda}$ as needed. By Theorem 4, the seed required is $\ell = O(\lambda \log(nM/\epsilon)(\log\log(nM/\epsilon))^2)$ as needed. The space and time required to compute each coordinate follows from Proposition 3 below. ∎

PROPOSITION 3. *In the setting of Lemma 7, if $G(y) = X' \in \{-M, \ldots, M\}^n$ is the output of G on random seed $y \in \{0,1\}^\ell$, then each coordinate X'_i can be computed in $O(\ell)$-space and in $\tilde{O}(1)$ time, where $\tilde{O}$ hides poly$(\log(nM))$ factors.*

Proof. In order to analyze the space complexity and runtime needed to compute a coordinate X'_i, we must describe the PRG of Theorem 4. The Goplan–Kane–Meka PRG has three main components, which themselves use other PRGs such as Nisan's PRG as subroutines. Recall that the PRG generates a pseudouniform element from $X \sim [M]^n$ that fools a class of Fourier shapes $f : [M]^n \to \mathbb{C}$ on truly uniform input in $[M]^n$. Note that because of the definition of a Fourier shape, if we wish to sample from a distribution $X \sim \mathcal{D}$ over $\{-M, \ldots, M\}^n$ that is not uniform, but such that X_i can be sampled with $\log(M')$-bits, we can first fool Fourier shapes $f' : [M']^n \to \mathbb{C}$, and then use a function $Q : [M'] \to \{-M, \ldots, M\}$ which samples $X_i \sim \mathcal{D}$ given $\log(M')$ uniformly random bits. We then fool Fourier shapes $F = \prod_{j=1}^n f'_j(x) = \prod_{j=1}^n f_j(Q(y))$, where x, y are uniform, and thus $Q(y) \sim \mathcal{D}$. Thus it will suffice to fool (M', n)-Fourier shapes on uniform distributions. For simplicity, for the most part we will omit the parameter ϵ in this discussion.

The three components of the PRG appear in sections 5, 6, and 7 of [GKM18], respectively. In this proof, when we write section x we are referring to the corresponding section of [GKM18]. They consider two main cases: one where the function f has *high variance* (for some notion of variance), and one where it has low variance. The PRGs use two main pseudorandom primitives, δ-biased and k-wise independent hash function families. Formally, a family $\mathcal{H} = \{h : [n] \to [M]\}$ is said to be δ-biased if for all $r \leq n$ distinct indices $i_1, \ldots, i_r \in [n]$ and $j_1, \ldots, j_r \in [M]$ we have

$$\mathbf{Pr}_{h \sim \mathcal{H}}[h(i_1) = j_1 \wedge \cdots \wedge h(i_r) = j_r] = \frac{1}{M^r} \pm \delta.$$

The function is said to be k-wise independent if it holds with $\delta = 0$ for all $r \leq k$. It is standard that k-wise independent families can be generated by taking a polynomial of degree k over a suitably large finite field (requiring space $O(k \log(Mn))$). Furthermore, a value $h(i)$ from a δ-biased family can be generated by taking products of two $O(\log(n/\delta))$-bit integers over a suitable finite field [Kop13] (requiring space $O(\log(n/\delta))$). So in both cases, computing a value $h(i)$ can be done in space and time that is linear in the space required to store the hash functions (or $O(\log(n/\delta))$-bit integers). Thus, any nested sequence of such hash functions used to compute a given coordinate X'_i can be carried out in space linear in the size required to store all the hash functions.

Now the first PRG (section 5 [GKM18]) handles the high variance case. The PRG first subsamples the n coordinates at $\log(n)$ levels using a pairwise hash function (note that a 2-wise permutation is used in [GKM18], which reduces to computation of a 2-wise hash function). In each level S_j of subsampling, it uses $O(1)$-wise independent hash functions to generate the coordinates $X_i \in S_j$. So if we want to compute a value X_i, we can carry out one hash function computation $h(i)$ to determine j such that $X_i \in S_j$ and then carry out another hash function computation $h_j(i) = X_i$. Instead of using $\log(n)$ independent hash functions h_j, each of size $O(\log(nM))$, for each of the buckets S_j, they derandomize this with the PRG of Nisan and Zuckerman [NZ96] to use a single seed of length $O(\log n)$. Now the PRG of Nisan and Zuckerman can be evaluated online, in the sense that it reads its random bits in a stream and writes its pseudorandom output on a one-way tape, and runs in space linear in the seed required to store the generator itself (see Definition 4 of [NZ96]). Such generators are composed to yield the final PRG of Theorem 2 [NZ96], but by Lemma 4 of the paper, such online generators are composable. Thus the entire generator of [NZ96] is online, and so any substring of the pseudorandom output can be computed in space linear in the seed of the generator by a single pass over the random input. Moreover, by Theorem 1 of [NZ96] in the setting of [GKM18], such a substring can be computed in $\tilde{O}(1)$ time, since it is only generating $\tilde{O}(1)$ random bits to begin with.

On top of this, the PRG of section 5 [GKM18] first splits the coordinates $[n]$ via a limited independence hash function into $\text{poly}(\log(1/\epsilon))$ buckets and applies the algorithm described above on each. To do this second layer of bucketing and not need fresh randomness for each bucket, they use Nisan's PRG [Nis92] with a seed of length $\log(n)\log\log(n)$. Now any bit of Nisan's PRG can be computed by several nested hash function computations, carried out in space linear in the seed required to store the PRG. Thus any substring of Nisan's can be computed in space linear in the seed and time $\tilde{O}(1)$. Thus to compute X_i', we first determine which bucket it hashes to, which involves computing random bits from Nisan's PRG. Then we determine a second partitioning, which is done via a 2-wise hash fucntion, and finally we compute the value of X_i' via an $O(1)$-wise hash function, where the randomness for this hash function is stored in a substring output by the PRG of [NZ96]. Altogether, we conclude that the PRG of section 5 [GKM18] is such that value X_i' can be computed in space linear in the seed length and $\tilde{O}(1)$ time.

Next, in section 6 of [GKM18], another PRG is introduced which reduces the problem to the case of $M \leq poly(n)$. Assuming a PRG G_1 is given which fools (M, n)-Fourier shapes, they design a PRG G_2 using G_1 which fools (M^2, n)-Fourier shapes. Applying this $O(\log\log(M))$ times reduces to the case of $m \leq n^4$. The PRG is as follows. Let $G_1, \ldots, G_t$ be the iteratively composed generators, where $t = O(\log\log(M))$. To compute the value of $(G_i)_j \in [M]$, where $(G_i)_j$ is the jth coordinate of $G_i \in [M]^n$, the algorithm first implicitly generates a matrix $Z \in [M]^{\sqrt{M}\times M}$. An entry $Z_{p,q}$ is generated as follows. First one applies a k-wise hash function $h(q)$ (for some k) and uses the $O(\log M)$-bit value of $h(q)$ as a seed for a second 2-wise indepedent hash function $h'_{h(q)}$. Then $Z_{p,q} = h'_{h(q)}(p)$. Thus within a column q of Z, the entries are 2-wise independent, and separate columns of Z are k-wise independent. This requires $O(k\log M)$-space to store, and the nested hash functions can be computed in $O(k\log M)$-space. Thus computing $Z_{i,j}$ is done in $\tilde{O}(1)$ time and space linear in the seed length. Then we set $(G_i)_j = Z_{(G_{i-1})_j, j}$ for each $j \in [n]$. Thus $(G_i)_j$ only depends on $(G_{i-1})_j$, and the random seeds stored for two hash functions to evaluate entries of Z. So altogether, the final output coordinate $(G_t)_j$ can be computed

in space linear in the seed length required to store all required hash functions, and in time $\tilde{O}(1)$. Note importantly that the recursion is linear, in the sense that computing $(G_i)_j$ involves only one query to compute $(G_i)_{j'}$ for some j'.

Next, in section 7 of [GKM18], another PRG is introduced for the *low-variance case*, which reduces the size of n to $\sqrt{n}$, but blows up m polynomially in the process. Formally, it shows given a PRG G_1' that fools $(\text{poly}(n), \sqrt{n})$ Fourier shapes, one can design a PRG G_2' that fools $O(M,n)$-Fourier shapes with $M < n^4$ (here the $\text{poly}(n)$ can be much larger than n^4). To do so, the PRG first hashes the n coordinates into $\sqrt{n}$ buckets k-wise independently and then in each bucket uses k-wise independence to generate the value of the coordinate. A priori, this requires $\sqrt{n}$ independent seeds for the hash function in each of the buckets. To remove this requirement, it uses G_1' to generate the $\sqrt{n}$ seeds required from a smaller seed. Thus to compute a coordinate i of G_2', simply evaluate a k-wise independent hash function on i to determine which bucket $j \in [\sqrt{n}]$ the item i is hashed into. Then evaluate $G_1'(j)$ to obtain the seed required for the k-wise hash function h_j, and the final result is given by $h_j(i)$. Note that this procedure only requires one query to the prior generator G_1'. The space required to do so is linear in the space required to store the hash functions, and the space required to evaluate a coordinate of the output of G_1', which will be linear in the size used to store G_1' by induction.

Finally, the overall PRG composes the PRG from sections 6 and 7 to fool larger n, M in the case of low variance. Suppose we are given a PRG G_0 which fools $(M'', \sqrt{n'})$-Fourier shapes for some $M'' < (n')^2$. We show how to construct a PRG G_1 which fools (M', n')-Fourier shapes for any $M' \leq (n')^4$. Let G^{6+7} be the PRG obtained by first applying the PRG from section 6 on G_0 as an initial point, which gives a PRG that fools $(\text{poly}(n'), \sqrt{n'})$-Fourier shapes, and then applying the PRG from section 7 on top, which now fools (M', n')-Fourier shapes (with low variance). Let G^5 be the generator from section 5 which fools (M', n')-Fourier shapes with high variance. The final algorithm for fooling the class of all (M', n')-Fourier shapes given G_0 computes a generator G_1 such that the ith coordinate is $(G_1)_i = (G^{6+7})_i \oplus (G^5)_i$, where $\oplus$ is addition mod M'. This allows one to simultaneously fool high and low variance Fourier shapes of the desired M', n'. If $M > (n')^4$, one can apply the PRG for section 6 one last time on top of G_1 to fool arbitrary M. Thus if for any i, the ith coordinate of G_{6+7} and G_5 can be composed in $\tilde{O}(1)$ time and space linear in the size required to store the random seed, then so can G_i. Thus going from G_0 to G_1 takes a generator that fools $(M'', \sqrt{n'})$ to (M', n')-Fourier shapes, and similarly we can compose this to design a G_2 that fools $(M', (n')^2)$-Fourier shapes. Composing this $t = O(\log \log n)$-times, we obtain G_t which fools $O(M,n)$ Fourier shapes for any M, n. As a base case (to define the PRG G_0), the PRG of [NZ96] is used, which we have already discussed can be evaluated on-line in space linear in the seed required to store it and time polynomial in the length of the seed.

Now we observe an important property of this recursion. At every step of the recursion, one is tasked with computing the jth coordinate output by some PRG for some j, and the result will depend *only* on a query for the j'th coordinate of another PRG for some j' (as well as some additional values which are computed using the portion of the random seed dedicated to this step in the recursion). Thus at every step of the recursion, only one query is made for a coordinate to a PRG at a lower level of the recursion. Thus the recursion is linear, in the sense that the computation path has only L nodes instead of 2^L (which would occur if two queries to coordinate j', j'' were made to a PRG in a lower level). Since at each level of recursion, computing G^{6+7} itself uses $O(\log \log(nM))$ levels of recursion, and also has the property that

each level queries the lower level at only one point, it follows that the total depth of the recursion is $O((\log\log(nM))^2)$. At each point, to store the information required for this recursion on the stack requires only $O(\log(nM))$-bits of space to store the relevant information identifying the instance of the PRG in the recursion, along with its associated portion of the random seed. Thus the total space required to compute a coordinate via these $O(\log\log(nM))^2)$ recursions is $O(\log(nM)(\log\log nM)^2)$, which is linear in the seed length. Moreover, the total runtime is $\tilde{O}(1)$, since each step of the recursion requires at most $\tilde{O}(1)$ time. ∎

We use the prior technique to derandomize a wide class of linear sketches $A \cdot f$ such that the entries of A are independent and can be sampled using $O(\log(n))$-bit, and such that the behavior of the algorithm only depends on the sketch Af. It is well known that there are strong connections between turnstile streaming algorithms and linear sketches, insofar as practically all turnstile streaming algorithms are in fact linear sketches. The equivalence of turnstile algorithms and linear sketches has even been formalized [LNW14], with some restrictions. Our results show that all such sketches that use independent, efficiently sampled entries in their sketching matrix A can be derandomized with our techniques. As an application, we derandomize the count-sketch variant of Minton and Price [MP14], a problem which to the best of the authors' knowledge was hitherto open.

LEMMA 8. *Let* `ALG` *be any streaming algorithm which, on stream vector* $f \in \{-M, \dots, M\}^n$ *for some* $M = poly(n)$*, stores only a linear sketch* $A \cdot f$ *such that the entries of the random matrix* $A \in \mathbb{R}^{k \times n}$ *are i.i.d., and can be sampled using* $O(\log(n))$*-bits. Fix any constant* $c \geq 1$*. Then* `ALG` *can be implemented using a random matrix* A' *using* $O(k\log(n)(\log\log n)^2)$ *bits of space, such that for every vector* $y \in \mathbb{R}^k$ *with entrywise bit-complexity of* $O(\log(n))$*,*

$$\left|\mathbf{Pr}\Big[Af = y\Big] - \mathbf{Pr}\Big[A'f = y\Big]\right| < n^{-ck}.$$

Proof. We can first scale all entries of the algorithm by the bit complexity so that each entry in A is a $O(\log(n))$-bit integer. Then by Lemma 7, we can store the randomness needed to compute each entry of A' with $O(k\log(n)(\log\log n)^2)$-bits of space, such that A' n^{-ck}-fools the class of all $O(k)$-halfspace testers, in particular the one which checks, for each coordinate $i \in [k]$, whether both $(A'f)_i < y + 1$ and $(A'f)_i > y_i - 1$, and accepts only if both hold for all $i \in [k]$. By Proposition 3, the entries of A' can be computed in space linear in the size of the random seed required to store A'. Since we have scaled all values to be integers, n^{-ck} fooling this tester is equivalent to the theorem statement. Note that the test $(A'f)_i < y + 1$ can be made into a half-space test as follows. Let $X^i \in \mathbb{R}^{nk}$ be the vector such that $X^i_{j+(i-1)n} = f_j$ for all $j \in [n]$ and $X^i_j = 0$ otherwise. Let $\mathtt{vec}(A) \in \mathbb{R}^{nk}$ be the vectorization of A. Then $(Af)_i = \langle \mathtt{vec}(A), X^i \rangle$, and all the entries of $\mathtt{vec}(A)$ are i.i.d., which allows us to make the stated constraints into the desired half-space constraints. ∎

Observe that the above lemma derandomized the linear sketch Af by writing each coordinate $(Af)_i$ as a linear combination of the random entries of $\mathtt{vec}(A)$. Note, however, that the above proof would hold if we added the values of any $O(k)$ additional linear combinations $\langle X_j, \mathtt{vec}(A)$ to the lemma, where each $X_j \in \{-M, \dots, M\}^{kn}$. This will be useful, since the behavior of some algorithms, for instance, count-sketch, may depend not only on the sketch Af but also on certain values or linear combinations of values within the sketch A. This is formalized in the following corollary.

COROLLARY 3. *Let the entries of $A \in \mathbb{R}^{k\times n}$ be drawn i.i.d. from a distribution which can be sampled using $O(\log n)$-bits, and let $\texttt{vec}(A) \in \mathbb{R}^{nk}$ be the vectorization of A. Let $X \in \mathbb{R}^{t\times nk}$ be any fixed matrix with entries contained within $\{-M,\dots,M\}$, where $M = poly(n)$. Then there is a distribution over random matrices $A' \in \mathbb{R}^{k\times n}$ which can be generated and stored using $O(t\log(n)(\log\log n)^2)$ bits of space, such that for every vector $y \in \mathbb{R}^t$ with entrywise bit-complexity of $O(\log(n))$,*

$$\left|\mathbf{Pr}\Big[X\cdot\texttt{vec}(A) = y\Big] - \mathbf{Pr}\Big[X\cdot\texttt{vec}(A') = y\Big]\right| < n^{-ct}.$$

Proof. The proof is nearly identical to Lemma 8, where we first scale entries to be $O(\log(n))$-bit integers, and then apply two half-space tests to each coordinate of $X\cdot\texttt{vec}(A')$. ∎

THEOREM 5. *Let ALG be any streaming algorithm which, on stream vector $f \in \{-M,\dots,M\}^n$ and fixed matrix $X \in \mathbb{R}^{t\times nk}$ with entries contained within $\{-M,\dots,M\}$, for some $M = poly(n)$, outputs a value that only depends on the sketches $A\cdot f$ and $X\cdot\texttt{vec}(A)$. Assume that the entries of the random matrix $A \in \mathbb{R}^{k\times n}$ are i.i.d. and can be sampled using $O(\log(n))$-bits. Let $\sigma : \mathbb{R}^k \times \mathbb{R}^t \to \{0,1\}$ be any tester which measures the success of ALG, namely, $\sigma(Af, X\cdot\texttt{vec}(A)) = 1$ whenever ALG succeeds. Fix any constant $c \geq 1$. Then ALG can be implemented using a random matrix A' using a random seed of length $O((k+t)\log(n)(\log\log n)^2)$, such that*

$$\left|\mathbf{Pr}\Big[\sigma(Af, X\cdot\texttt{vec}(A)) = 1\Big] - \mathbf{Pr}\Big[\sigma(A'f, X\cdot\texttt{vec}(A')) = 1\Big]\right| < n^{-c(k+t)}$$

and such that each entry of A' can be computed in time $\tilde{O}(1)$ and using working space linear in the seed length.

Proof. As in Lemma 8, we first scale all entries of the algorithm by the bit complexity so that each entry in A is a $O(\log(n))$-bit integer. Then there is a $M' = \text{poly}(n)$ such that each entry of $A\cdot f$ and $X\cdot\texttt{vec}(A)$ will be a integer of magnitude at most M'. First note that the sketch $A\cdot f$ and $X\cdot\texttt{vec}(A)$ can be written as one linear sketch $X_0\cdot\texttt{vec}(A)$, where $X_0 \in \mathbb{R}^{k+t\times kn}$. Then σ can be written as a function $\sigma : \mathbb{R}^{k+t} \to \{0,1\}$ evaluated on $\sigma(X_0\cdot\texttt{vec}(A))$. Let $S = \{y \in \{-M',\dots,M'\}^{k+t} \mid \sigma(y) = 1\}$. Then by Corollary 3, we have

$$|\mathbf{Pr}[X_0\cdot\texttt{vec}(A) = y] - \mathbf{Pr}[X_0\cdot\texttt{vec}(A') = y]| < n^{-c(k+t)}$$

for all $y \in S$. Taking c sufficiently large, and noting $|S| = n^{-O(k+t)}$, we have $\mathbf{Pr}[\sigma(X_0\cdot\texttt{vec}(A)) = 1] = \sum_{y\in S}\mathbf{Pr}[X_0\cdot\texttt{vec}(A) = y] = \sum_{y\in S}(\mathbf{Pr}[X_0\cdot\texttt{vec}(A') = y] \pm n^{-c(k+t)}) = \mathbf{Pr}[\sigma(X_0\cdot\texttt{vec}(A')) = 1] + n^{-O(k+t)}$ as desired. The final claim follows from Proposition 3. ∎

5.2.3. Derandomizing the count-sketch of Minton and Price. We now show how this general derandomization procedure can be used to derandomize the count-sketch variant of Minton and Price [MP14]. Our discussion will utilize the notation for count-sketch as defined in section 2.1. Minton and Price's analysis shows improved concentration bounds for count-sketch when the random signs $g_i(k) \in \{1,-1\}$ are fully independent. They demonstrate that in this setting, if $y \in \mathbb{R}^n$ is the count-sketch estimate of a stream vector f, where the count-sketch table A has k columns and d rows, then for any $t \leq d$ and index $i \in [n]$ we have

$$\mathbf{Pr}\Big[(f_i - y_i)^2 > \frac{t}{d}\frac{\|f_{\text{tail}(k))}\|_2^2}{k}\Big] \leq 2e^{-\Omega(t)}.$$

Notice that by setting $t = d = \Theta(\log 1/\delta)$, one recovers the standard count-sketch result of Theorem 1. However, in order to apply this algorithm in $o(n)$ space, one must first derandomize it from using fully independent random signs, which are not required for the original count-sketch of [CCFC02a]. To the best of the authors' knowledge, the best known derandomization procedure was a black-box application of Nisan's PRG which results in $O(\epsilon^{-2}\log^3(n))$-bits of space when $k = O(1/\epsilon^2)$ and $d = O(\log n)$. Due to this $\log n$ blow-up in the space, the guarantees of this count-sketch variant, if derandomized with Nisan's PRG, are strictly worse than using the original count sketch of [CCFC02a]. Our derandomization, in contrast, demonstrates a strong improvement on this, obtaining the same bounds with an $(\log\log n)^2$ instead of an $\log n$ factor blow-up. For the purpose of the theorem, we replace the notation $1/\epsilon^2$ with k (the number of columns of count-sketch up to a constant).

THEOREM 6. *The count-sketch variant of* [MP14] *can be implemented so that if* $A \in \mathbb{R}^{d\times k}$ *is a count-sketch table, then for any* $t \le d$ *and index* $i \in [n]$ *we have*

$$\mathbf{Pr}\Big[(f_i - y_i)^2 > \frac{t}{d}\frac{\|f_{\mathrm{tail}(k))}\|_2^2}{k}\Big] \le 2e^{-\Omega(t)}$$

and such that the total space required is $O(kd\log(n)(\log\log n)^2)$.

Proof. We first remark that the following modifcation to the count-sketch procedure does not affect the analysis of [MP14]. Let $A \in \mathbb{R}^{d\times k}$ be a $d \times k$ count-sketch matrix. The modification is as follows: instead of each variable $h_i(\ell)$ being uniformly distributed in $\{1, 2, \ldots, k\}$, we replace them with variables $h_{i,j,\ell} \in \{0, 1\}$ for $(i, j, \ell) \in [d] \times [k] \times [n]$, such that $h_{i,j,\ell}$ are all i.i.d. and equal to 1 with probability $1/k$. We also let $g_{i,h,\ell} \in \{1, -1\}$ be i.i.d. Rademacher variables (1 with probability 1/2). Then $A_{i,j} = \sum_{\ell=1}^{n} f_\ell g_{i,j,\ell} h_{i,j,\ell}$, and the estimate y_ℓ of f_ℓ for $\ell \in [n]$ is given by

$$y_\ell = \mathrm{median}\{g_{i,j,\ell}A_{i,j} \mid h_{i,j,\ell} = 1\}.$$

Thus the element f_ℓ can be hashed into multiple buckets in the same row of A, or even be hashed into none of the buckets in a given row. By Chernoff bounds, $|\{g_{i,j,\ell}A_{i,j} \mid h_{i,j,\ell} = 1\}| = \Theta(d)$ with high probability for all $\ell \in [n]$. Observe that the marginal distribution of each bucket is the same as the count-sketch used in [MP14], and moreover seperate buckets are fully independent. The key property used in the analysis of [MP14] is that the final estimator is a median over estimators whose error is independent and symmetric, and therefore the bounds stated in the theorem still hold after this modification [Pri18].

Given this, the entire sketch stored by the streaming algorithm is $B \cdot f$, where

$$B_j = \begin{cases} 1 & \text{with prob } \frac{1}{2k}, \\ -1 & \text{with prob } \frac{1}{2k}, \\ 0 & \text{otherwise.} \end{cases}$$

Thus the entries of B are i.i.d. and can be sampled with $O(\log(k)) \le O(\log(n))$ bits, and $\mathtt{vec}(A) = B \cdot f$, where $\mathtt{vec}(A)$ is the vectorization of the count-sketch table A. Here $B \in \mathbb{R}^{dk\times n}$.

Now note that for a fixed i, to test the statement that $(f_i - y_i)^2 > \frac{t}{d}\frac{\|f_{\mathrm{tail}(k))}\|_2^2}{k}$, one needs to know both the value of the sketch Bf, in addition to the value of the ith column of B, since the estimate can be written as $y_i = \mathrm{median}_{j\in[kd],B_{j,i}\neq 0}\{B_{j,i} \cdot (Bf)_j\}$. Note that the ith column of B (which has kd entries) can simply be written

as a sketch of the form $X \cdot \mathtt{vec}(B)$, where $X \in \mathbb{R}^{kd \times dkn}$ is a fixed matrix such that $X \cdot \mathtt{vec}(B) = B_i$, so we also need to store $X \cdot \mathtt{vec}(B)$. Thus by Theorem 5, the algorithm can be derandomized to use $O(kd \log(n)(\log\log n)^2)$ bits of space, and such that for any $t \leq d$ and any $i \in [n]$ we have $\mathbf{Pr}[(f_i - y_i)^2 > \frac{t}{d}\frac{\|f_{\mathrm{tail}(k))}\|_2^2}{k}] \leq 2e^{-\Omega(t)} \pm n^{-\Omega(dk)}$. ∎

5.2.4. Derandomizing the L_p sampling algorithm. We now introduce the notation which will be used in our derandomization. Our L_p sampler uses two sources of randomness which we must construct PRGs for. The first, r_e, is the randomness needed to construct the exponential random variables t_i, and the second, r_c, is the randomness needed for the fully random hash functions and signs used in count-max. Note that r_e, r_c both require $\mathrm{poly}(n)$ bits by Lemma 6. From here on, we will fix any index $i \in [n]$. Our L_p sampler can then be thought of as a tester $\mathcal{A}(r_e, r_c) \in \{0, 1\}$, which tests on inputs r_e, r_c, whether the algorithm will output $i \in [n]$. Let $G_1(x)$ be Nisan's PRG, and let $G_2(y)$ be the half-space PRG. For two values $b, c \in \mathbb{R}$, we write $a \sim_\epsilon b$ to denote $|a - b| < \epsilon$. Our goal is to show that

$$\mathbf{Pr}_{r_e, r_c}\big[\mathcal{A}(r_e, r_c)\big] \sim_{n^{-c}} \mathbf{Pr}_{x,y}\big[\mathcal{A}(G_2(y), G_1(x))\big],$$

where x, y are seeds of length at most $O(\log^2 n(\log\log n)^2)$, and c is an arbitrarily large constant.

THEOREM 7. *A single instance of the algorithm L_p* `Sampler` *using* `Fast-Update` *as its update procedure can be derandomized using a random seed of length $O(\log^2(n)(\log\log n)^2)$, and thus can be implemented in this space. Moreover, this does not affect the time complexity as stated in Lemma* 6.

Proof. First note that by Lemma 6, we require $\tilde{O}(\nu^{-1})$ random bits for each $i \in [n]$, and thus we require a total of $\tilde{O}(n\nu^{-1}) = \mathrm{poly}(n)$ random bits to be generated. Since Nisan's PRG requires the tester to read its random input in a stream, we can use a standard reordering trick of the elements of the stream, so that all the updates to a given coordinate $i \in [n]$ occur at the same time (see [Ind06]). This does not affect the output distribution of our algorithm, since linear sketches do not depend on the ordering of the stream. Now let c' be the constant such that the algorithm L_p `Sampler` duplicates coordinates $n^{c'}$ times. In other words, the count-max is run on the stream vector $F \in \mathbb{R}^{n^{c'}}$, and let $N = n^{c'}$. Now, as above, we fix any index $i \in [N]$ and attempt to fool the tester which checks if, on a given random string, our algorithm would output i. For any fixed randomness r_e for the exponentials, let $\mathcal{A}_{r_e}(r_c)$ be the tester which tests if our L_p sampler would output the index i, where now the bits r_e are hard-coded into the tester, and the random bits r_c are taken as input and read in a stream. We first claim that this tester can be implemented in $O(\log(n))$-space.

To see this, note that $\mathcal{A}_{r_e}(r_c)$ must simply count the number of rows of count-max such that item i is hashed into the largest bucket (in absolute value) of that row, and output 1 if this number is at least $\frac{4d}{5}$, where d is the number of rows in count-max. To do this, $\mathcal{A}_{r_e}(r_c)$ can break r_c into d blocks of randomness, where the jth block is used only for the jth row of count-max. It can then fully construct the values of the counters in a row, one row at a time, reading the bits of r_c in a stream. To build a bucket, it looks at the first element of the stream, uses r_c to find the bucket it hashes to and the Gaussian scaling it gets, then adds this value to that bucket, and then continues with the next element. Note that since r_e is hardcoded into the tester, we

can assume the entire stream vector ζ is hardcoded into the tester. Once it constructs a row of count-max, it checks if i is in the largest bucket by absolute value, and increments a $O(\log(d))$-bit counter if so. Note that it can determine which bucket i hashes to in this row while reading off the block of randomness corresponding to that row. Then, it throws out the values of this row and the index of the bucket i hashed to in this row and builds the next row. Since each row has $O(1)$ buckets, $\mathcal{A}_{r_e}(r_c)$ only uses $O(\log(n))$-bits of space at a time. Then using $G_1(x)$ as Nisan's generator with a random seed x of length $O(\log^2(n))$-bits, we have $\mathbf{Pr}[\mathcal{A}_{r_e}(r_c)] \sim_{n^{-c_0}} \mathbf{Pr}[\mathcal{A}_{r_e}(G_1(x))]$, where the constant c_0 is chosen to be sufficiently larger than the constant c_1 in the n^{-c_1} additive error of our perfect sampler, as well as the constant c'. Moreover,

$$\begin{aligned}
\mathbf{Pr}\Big[\mathcal{A}(r_e, r_c)\Big] &= \sum_{r_e} \mathbf{Pr}\Big[\mathcal{A}_{r_e}(r_c)\Big]\mathbf{Pr}\Big[r_e\Big] \\
&= \sum_{r_e} \Big(\mathbf{Pr}\Big[\mathcal{A}_{r_e}(G_1(x))\Big] \pm n^{-c_0}\Big)\mathbf{Pr}\Big[r_e\Big] \\
&= \sum_{r_e} \mathbf{Pr}\Big[\mathcal{A}_{r_e}(G_1(x))\Big]\mathbf{Pr}\Big[r_e\Big] \pm \sum_{r_e} n^{-c_0}\mathbf{Pr}\Big[r_e\Big] \\
&\sim_{n^{-c_0}} \mathbf{Pr}\Big[\mathcal{A}(r_e, G_1(x))\Big].
\end{aligned}$$

Now fix any Nisan seed x and consider the tester $\mathcal{A}_{G_1(x)}(r_e)$, which on fixed count-max randomness $G_1(x)$ tests if the algorithm will output $i \in [n]$ on the random input r_e for the exponential variables. We first observe that it seems unlikely that $\mathcal{A}_{G_1(x)}(r_e)$ can be implemented in $\log(n)$ space while reading its random bits r_e in a stream. This is because each row of count-max depends on the same random bits in r_e used to construct the exponentials t_i, and thus it seems $\mathcal{A}_{G_1(x)}(r_e)$ would need to store all $\log^2(n)$ bits of count-max at once. However, we will now demonstrate that $\mathcal{A}_{G_1(x)}(r_e)$ is in fact a $\text{poly}(n)$ bounded $O(d)$-halfspace tester (as defined earlier in this section) where d is the number of rows of count-max, and therefore can be derandomized with the PRG of [GKM18]. By the Runtime & Random Bits analysis in Lemma 6, it suffices to take all random variables in the algorithm to be $O(\log(n))$-bit rational numbers. Scaling by a sufficiently large $\text{poly}(n)$, we can assume that $1/t_j^{1/p}$ is a discrete distribution supported on $\{-T, \dots, T\}$, where $T \leq \text{poly}(n)$ for a sufficiently large $\text{poly}(n)$. We can then remove all values in the support which occur with probability less than $\text{poly}(n)$, which only adds an n^{-c_0} additive error to our sampler. After this, the distribution can be sampled from with $\text{poly}(T) = \text{poly}(n)$ random bits, which is as needed for the setting of Lemma 7. Note that we can also apply this scaling to the Gaussians in count-max, so that they too are integers of magnitude at most $\text{poly}(n)$.

Given this, the distribution of the variables $1/t_j^{1/p}$ satisfy the conditions of Lemma 7, in particular being $\text{poly}(n)$-bounded; thus we must now show that $\mathcal{A}_{G_1(x)}(r_e)$ is indeed a $O(d)$-halfspace tester, with integer valued half-spaces bounded by $\text{poly}(n)$. First consider a given row of count-max, and let the buckets be B_1, B_2. WLOG i hashes into B_1, and we must check if $|B_1| > |B_2|$. Let g_j be the random count-max signs (as specified by $G_1(x)$), and let S_1, S_2 be the set of indices which hash to B_1 and B_t, respectively. We can run the following six half-space tests to test if $|B_1| > |B_2|$:

$$\sum_{j \in S_1} g_j f_j \left(\frac{1}{t_j^{1/p}}\right) > 0, \tag{2}$$

$$\sum_{j\in S_2} g_j f_j\left(\frac{1}{t_j^{1/p}}\right) > 0, \tag{3}$$

$$a_1 \sum_{j\in S_1} g_j f_j\left(\frac{1}{t_j^{1/p}}\right) + a_2 \sum_{j\in S_2} g_j f_j\left(\frac{1}{t_j^{1/p}}\right) > 0, \tag{4}$$

where a_1, a_2 range over all values in $\{1, -1\}^2$. The tester can decide whether $|B_1| > |B_2|$ by letting a_1 be the truth value (where -1 is taken as fail) of the first test 2 and a_2 the truth value of 3. It then lets $b_2 \in \{0, 1\}$ be the truth value of 4 on the resulting a_1, a_2 values, and it can correctly declare $|B_1| > |B_2|$ iff $b_2 = 1$. Thus for each pair of buckets, the tester uses six halfspace testers to determine if $|B_1| > |B_2|$, and so can determine if i is hashed to the max bucket with $O(1)$ halfspace tests. So $\mathcal{A}_{G_1(x)}(r_e)$ can test if the algorithm will output i by testing if i is hashed to the max bucket in a $4/5$ fraction of the d rows of count-max, using $O(d) = O(\log(n))$ halfspace tests. Note that by the scaling performed in the prior paragraphs, all coefficents of these half-spaces are integers of magnitude at most $\text{poly}(n)$. So by Lemma 7, the PRG $G_2(y)$ of [GKM18] fools $\mathcal{A}_{G_1(x)}(r_e)$ with a seed y of $O(\log^2(n)(\log\log n)^2)$-bits. So $\mathbf{Pr}[\mathcal{A}_{G_1(x)}(r_e)] \sim_{n^{-c_0}} \mathbf{Pr}[\mathcal{A}_{G_1(x)}(G_2(y))]$, and so by the same averaging argument as used for the Nisan PRG above, we have $\mathbf{Pr}[\mathcal{A}(r_e, G_1(x))] \sim_{n^{-c_0}} \mathbf{Pr}[\mathcal{A}(G_2(y), G_1(x))]$, and so $\mathbf{Pr}[\mathcal{A}(r_e, r_c)] \sim_{n^{-c_0}} \mathbf{Pr}[\mathcal{A}(G_2(y), G_1(x))]$ as desired. Now fixing any $i \in [n]$, let $\mathcal{A}'(r_e, r_c)$ be the event that the overall algorithm outputs the index i. In other words, $\mathcal{A}'_i(r_e, r_c) = 1$ if $\mathcal{A}_{i_j}(r_e, r_c) = 1$ for some $j \in [n^{c'-1}]$, where $\mathcal{A}_{i_j}(r_e, r_c) = 1$ is the event that count-max declares that i_j is the maximum in Algorithm L_p `Sampler`. Thus, the probability that the algorithm outputs a nonduplicated coordinate $i \in [n]$ is given by

$$\begin{aligned}
\mathbf{Pr}\Big[\mathcal{A}'_i(r_e, r_c)\Big] &= \sum_{j=1}^{n^{c'}} \mathbf{Pr}\Big[\mathcal{A}_{i_j}(r_e, r_c)\Big] \\
&= \sum_{j=1}^{n^{c'}} \mathbf{Pr}\Big[\mathcal{A}_{i_j}(G_2(y), G_1(x))\Big] \pm n^{-c_0} \\
&= \mathbf{Pr}\Big[\mathcal{A}'_i(G_2(y), G_1(x))\Big] \pm n^{-c_1},
\end{aligned} \tag{5}$$

where in the last line we set $c_0 > c' + c_1$, where recall c_1 is the desired additive error in our main sampler. In conclusion, replacing the count-max randomness with Nisan's PRG and the exponential random variable randomness with the half-space PRG $G_2(y)$, we can fool the algorithm which tests the output of our algorithm with a total seed length of $O(\log^2(n)(\log\log n)^2)$.

To show that the stated update time of Lemma 6 is not affected, we first remark that Nisan's PRG simply involves performing $O(\log(n))$ nested hash computations on a string of length $O(\log(n))$ in order to obtain any arbitrary substring of $O(\log(n))$ bits. Thus the runtime of such a procedure is $\tilde{O}(1)$ to obtain the randomness needed in each update of a coordinate $i \in [n^c]$. By Lemma 7, the PRG of [GKM18] requires $\tilde{O}(1)$ time to sample the $O(\log(n))$-bit string needed to generate an exponential, and moreover can be computed with working space linear in the size of the random seed. (Note that this is also true of Nisan's PRG, which just involves $O(\log(n))$-nested hash function computations.) Thus the update time is only blown up by a $\tilde{O}(1)$ factor, which completes the proof. ∎

COROLLARY 4. *For $p = 2$, the entire algorithm can be derandomized to run using $O(\log^3(n)\log(1/\delta))$-bits of space with failure probability of δ. For $p < 2$, the algorithm can be derandomized to run using $O(\log^3(n))$-bits of space with $\delta = 1/poly(n)$.*

Proof. We can simply derandomize a single instance of our sampling algorithm using Nisan's PRG as in Theorem 7, except that we derandomize all the randomness in the algorithm at once. Since such an instance requires $O(\log^2(n))$-bits of space, using Nisan's blows up the complexity to $O(\log^3(n))$. (The tester can simply simulate our entire algorithm in $O(\log^2(n))$-bits of space, reading the randomness in a stream by the reordering trick of [Ind06].) Since the randomness for separate parallel instances of the main sampling algorithm is disjoint and independent, this same $O(\log^2(n))$-bit tester can test the entire output of the algorithm by testing each parallel instance one by one, and terminating on the first instance that returns an index $i \in [n]$. Thus the same $O(\log^3(n))$-bit random seed can be used to randomize all parallel instances of our algorithm. For $p < 2$, we can run $O(\log(n))$ parallel instances to get $1/\text{poly}(n)$ failure probability in $O(\log^3(n))$-bits of space as stated. For $p = 2$, we can run $O(\log(n)\log(1/\delta))$ parallel repetitions needed to get δ failure probability using the same random string, for a total space of $O(\log^3(n)\log(1/\delta) + \log^3(n)) = O(\log^3(n)\log(1/\delta))$ as stated. As noted in the proof of Theorem 7, computing a substring of $O(\log(n))$-bits from Nisan's PRG can be done in $\tilde{O}(1)$ time and using space linear in the seed length, which completes the proof. ∎

5.3. Query time. We will now show the modifications to our algorithm necessary to obtain $\tilde{O}(1)$ query time. Recall that our algorithm maintains a count-max matrix A. Our algorithm then searches over all indices $i \in \mathcal{K}$ to check if i is hashed into the maximum bucket in a row of A at least a $4/5$ fraction of the time. Since $|\mathcal{K}| = \tilde{O}(n)$, running this procedure requires $\tilde{O}(n)$ time to produce an output on a given query. To avoid this and obtain $\tilde{O}(1)$ running time, we will utilize the heavy hitters algorithm of [LNNT16], which has a $\tilde{O}(1)$ update and query time, and which does not increase the complexity of our algorithm.

THEOREM 8 ([LNNT16]). *For any precision parameter $0 < \epsilon < 1/2$, given a general turnstile stream $x \in \mathbb{R}^n$ there is an algorithm, `ExpanderSketch`, which with probability $1 - n^{-c}$ for any constant c returns a set $S \subset [n]$ of size $S = O(\epsilon^{-2})$ which contains all indices i such that $|x_i| \geq \epsilon\|x\|_2$. The update time is $O(\log(n))$, the query time is $\tilde{O}(\epsilon^{-2})$, and the space required is $O(\epsilon^{-2}\log^2(n))$-bits.*

Using `ExpanderSketch` to speed up query time. The modifications to our main algorithm L_p `Sampler` with `Fast-Update` are as follows. We run our main algorithm as before, maintaining the same count-max data structures A. Upon initialization of our algorithm, we also initialize an instance `ExSk` of `ExpanderSketch` as in Theorem 8, with the precision parameter $\epsilon = 1/100$.

Now recall in our `Fast-Update` procedure, for each $i \in [n]$ we hash the top $K_i = O(\log(n))$ largest duplicates ζ_{i_j} corresponding to f_i individually and store the random variables $h_\ell(i_j)$ that determine which buckets in A they hash to. While processing updates to our algorithm at this point, we make the modification of *additionally* sending these top K_i items to `ExSk` to be sketched. More formally, we run `ExSk` on the stream $\zeta_{\mathcal{K}}$, where $\zeta_{\mathcal{K}}$ is the vector ζ projected onto the coordinates of $\mathcal{K}$. Since $K_i = \tilde{O}(1)$, this requires making $\tilde{O}(1)$ calls to update `ExSk` on different coordinates, which only increases our update time by an $\tilde{O}(1)$ additive term.

On termination, we obtain the set S containing all items ζ_i such that $i \in \mathcal{K}$ and $\zeta_i \geq (1/100)\|\zeta_{\mathcal{K}}\|_2$. Instead of searching through all coordinates of $\mathcal{K}$ to find a

maximizer, we simply search through the coordinates in S, which takes $\tilde{O}(|S|) = \tilde{O}(1)$ time. We now argue that the output of our algorithm does not change with these new modifications. We refer collectively to the new algorithm with these modifications as L_p `Sampler` with `Fast-Update` and `ExSk`, and the algorithm of section 5.1 as simply L_p `Sampler` with `Fast-Update`.

LEMMA 9. *For any constant $c > 0$, with probability $1 - n^{-100c}$ the algorithm L_p `Sampler` with `Fast-Update` and `ExSk` as described in this section returns the same output (an index $i \in [n]$ or `FAIL`) as L_p `Sampler` using `Fast-Update` but without `ExSk`. The space and update time are not increased by using `ExSk`, and the query time is now $\tilde{O}(1)$.*

Proof. We condition on the event that S contains all items i such that $i \in \mathcal{K}$ and $|\zeta_i| \geq 1/100\|\zeta_{\mathcal{K}}\|_2$, which occurs with probability $1 - n^{-100c}$ by Theorem 8. Since L_p `Sampler` already uses at least $O(\log^2(n))$ bits of space, the additional $O(\log^2(n))$ bits of overhead required to run an instance `ExSk` of `ExpanderSketch` with sensitivity parameter $\epsilon = 1/100$ does not increase the space complexity. Furthermore, as mentioned above, the update time is blown up by a factor of $\tilde{O}(1)$, since we make $K_i = \tilde{O}(1)$ calls to update `ExSk`, which has an update time of $\tilde{O}(1)$ by Theorem 8. Furthermore, our algorithm does not require any more random bits, as it only uses `ExpanderSketch` as a subroutine, and thus no further derandomization is required. Thus the complexity guarantees of Lemma 6 are unchanged. For the query time, we note that obtaining S requires $\tilde{O}(1)$ time (again by Theorem 8), and querying each of the $|S| = O(1)$ items in our count-max A requires $\tilde{O}(1)$ time. To complete the proof, we now consider the output of our algorithm. Since we are searching through a strict subset $S \subset [n^c]$, it suffices to show that if the original algorithm outputted an $i_j \in [n^c]$, then so will we. As argued in Lemma 6, such a coordinate must be contained in $\mathcal{K}$. By Corollary 1, we must have $|\zeta_{i_j}| > \frac{1}{100}\|\zeta\|_2 \geq \frac{1}{100}\|\zeta_{\mathcal{K}}\|_2$ with probability $1 - n^{-100c}$ (scaling c by 100 here), and thus $i_j \in S$, which completes the proof. ∎

6. Estimating the frequency of the sampled coordinate. In this section, we will show how, conditioned on our algorithm L_p `Sampler` returning a sampled index $i \in [n]$, we can obtain an estimate $\tilde{f}_i = (1 \pm \epsilon)f_i$ with probability $1 - \delta_2$. We now describe how to do this. Our algorithm, in addition to the count-max matrix A used by L_p `Sampler`, stores a count-sketch matrix A' with $d' = O(\log(1/\delta_2))$ rows and $O(\gamma) = O(\min\{\epsilon^{-2}, \epsilon^{-p}\log(\frac{1}{\delta_2})\})$ columns. Recall in our `Fast-Update` procedure, for each $i \in [n]$ we hash the top $K_i = O(\log(n))$ largest duplicates ζ_{i_j} corresponding to f_i individually into A, and store the random variables $h_\ell(i_j)$ that determine which buckets in A they hash to. Thus if count-max outputs an $i_j \in [n^c]$ we know that $i_j \in \mathcal{K}$, where $\mathcal{K} = \cup_{i\in[n]} \cup_{j=1}^{K_i} \{i_j\}$ as in section 5 (since our algorithm only searches through $\mathcal{K}$ to find a maximizer). Thus it suffices to run the count-sketch instance A' on the stream $\zeta_{\mathcal{K}}$, where $\zeta_{\mathcal{K}}$ is the vector ζ with the coordinates not in $\mathcal{K}$ set to 0. Since $K_i = \tilde{O}(1)$, we perform at most $\tilde{O}(1)$ updates to count-sketch at every step in the stream. This requires making $\tilde{O}(1)$ calls to update count-sketch on each stream update, which only increases our update time by an $\tilde{O}(1)$ additive term.

Now if L_p `Sampler` returns $i_j \in [n^c]$ (corresponding to some duplicate i_j of i), then we must have $i_j \in \mathcal{K}$. Thus we can query A' for a value $\tilde{y}_{i_j}$ such that $|\tilde{y}_{i_j} - \zeta_{i_j}| < \sqrt{1/\gamma}\|\zeta_{\mathrm{tail}(\gamma)}\|_2$ with probability $1 - \delta_2$ by Theorem 1. Furthermore, since $i_j \in \mathcal{K}$, we can compute the value I_k such that $I_k = (\mathtt{rnd}_\nu(1/t_{i_j}))$ by simulating the `Fast-Update` procedure on an update to i. We will argue that the estimate

$\tilde{f} = \tilde{y}_{i_j}(\mathtt{rnd}_\nu(1/t_{i_j}^{1/p}))^{-1}$ satisfies $\tilde{f} = (1 \pm \epsilon)f_i$. Putting this together with Theorem 2, we will obtain the following result.

THEOREM 9. *There is an algorithm $\mathcal{A}$ which, on a general turnstile stream f, outputs $i \in [n]$ with probability $|f_i|^p/\|f\|_p^p(1\pm\nu)+O(n^{-c})$ and outputs* FAIL *with probability at most δ_1. Conditioned on outputting some $i \in [n]$, $\mathcal{A}$ will then output $\tilde{f}$ such that $\tilde{f} = (1 \pm \epsilon)f_i$ with probability $1 - \delta_2$. The space required is $O((\log^2(n)(\log\log n)^2 + \beta\log(n)\log(1/\delta_2))\log(1/\delta_1))$ for $p \in (0,2)$ and $O((\log^3(n) + \epsilon^{-2}\log^2(n)\log(1/\delta_2))\log(1/\delta_1))$ for $p = 2$, where $\beta = \min\{\epsilon^{-2}, \epsilon^{-p}\log(\frac{1}{\delta_2})\}\}$. The update time is $\tilde{O}(\nu^{-1})$ and the query time is $\tilde{O}(1)$.*

Proof. We first consider the complexity. The first term in each of the upper bounds follows from Theorem 2, as well as the $\log(1/\delta_1)$ term which comes from repeating the entire algorithm $\log(1/\delta_1)$ times for $p < 2$ and $\log(n)\log(1/\delta_1)$ times for $p = 2$. The second term in the space bound results from storing the $d' \times \gamma$ count-sketch table A', which is $O(\gamma\log(n)\log(1/\delta_2))$ as stated. Moreover, the update time for the new data structure is at most $\tilde{O}(1)$, since the only additional work we do on each update is to hash $K_i = O(\log(n))$ items into $d' = O(\log(n))$ rows of A'. Furthermore, the query time just requires computing a median of $O(\log(n))$ entries of A'. Each of these actions is $\tilde{O}(1)$ time in the unit cost RAM model, so the additional update and query time is $\tilde{O}(1)$. The remaining $\tilde{O}(\nu)$ update time follows from Lemma 6.

For correctness, note that if L_p `Sampler` does not fail and instead outputs $i_j \in [n^c]$, we know that $|\zeta_{i_j}| > 1/100\|\zeta\|_2$. Furthermore, we have $|\tilde{y}_{i_j} - \zeta_{i_j}| < \sqrt{1/\gamma}\|\zeta_{\mathrm{tail}(\gamma)}\|_2 \leq \sqrt{1/\gamma}\|\zeta\|_2$ with probability $1-\delta_2$, so setting $\gamma = \Theta(1/\epsilon^2)$ sufficiently large, it follows that $\tilde{y}_{i_j} = (1 \pm O(\epsilon))\zeta_{i_j}$. Then $\tilde{y}_{i_j}(\mathtt{rnd}_\nu(1/t_{i_j}^{1/p}))^{-1} = (1 \pm \epsilon)f_i$ follows immediately from the fact that $f_i = \zeta_{i_j}(\mathtt{rnd}_\nu(1/t_{i_j}^{1/p}))^{-1}$ (and a rescaling of ϵ by a constant). This shows that $O(\epsilon^{-2})$ bits is always an upper bound for the value of $\gamma = \Theta(\beta)$ needed for $p \in (0,2]$.

To show the other upper bound in the definition of β (for cases when $p < 2$), first define $T_\gamma \subset [n^c]$ as the set of $n^c - \gamma$ smallest coordinates (in absolute value) of z. In other words $z_{T_\gamma} = z_{\mathrm{tail}(\gamma)}$, where for any set $S \subseteq [n^c]$ z_S denotes z projected onto the coordinates of S. Note that if S is any set of size $n^c - s$ and $v \in \mathbb{R}^{n^c}$ any vector, we have $\|v_{\mathrm{tail}(s)}\|_2 \leq \|v_S\|_2$. Then by Proposition 1, using the fact that $\zeta_i = (1 \pm O(\nu))z_i$ for all $i \in [n^c]$, we have $\|\zeta_{\mathrm{tail}(\gamma)}\|_2 \leq \|\zeta_{T_\gamma}\|_2 \leq 2\|z_{\mathrm{tail}(\gamma)}\|_2 = O(\|F\|_p(\gamma)^{-1/p+1/2})$ for $p < 2$ with probability $1 - O(e^{-\gamma}) > 1 - \delta_2$, where now we are setting $\gamma = \Theta(\max\{\epsilon^{-p}, \log(1/\delta_2)\})$. Condition on this now. Then we obtain error $|\tilde{y}_{i_j} - \zeta_{i_j}| < \sqrt{1/\gamma}\|\zeta_{\mathrm{tail}(\gamma)}\|_2 = O(\|F\|_p\gamma^{-1/p}) = O(\epsilon(\log(1/\delta_2))^{-1/p}\|F\|_p)$ from our second count-sketch A'. Now $z_{D(1)} = \|F\|_p/E_1^{1/p}$, which is at least $\Omega(\|F\|_p/(\log(1/\delta_2))^{1/p})$ with probability greater than $1 - \delta_2$ using the pdf of an exponential. Conditioned on this, the error from our second count-sketch A' gives, in fact, a $(1 \pm O(\epsilon))$ relative error approximation of ζ_{i_j}, which is the desired result. Note that we conditioned only on our count-sketch giving the desired $|\tilde{y}_{i_j} - \zeta_{i_j}| < \sqrt{1/\gamma}\|\zeta_{\mathrm{tail}(\gamma)}\|_2$ error, on $\|z_{\mathrm{tail}(\gamma)}\|_2 = O(\|F\|_p(\gamma)^{-1/p+1/2})$, and on $E_1 = O(\log(1/\delta_2))$, each of which holds with probability at least $1 - O(\delta_2)$, so the Theorem follows after a union bound. ∎

7. Lower bounds. In this section, we obtain a lower bound for providing relative error approximations of the frequency of a sampled item. Our lower bound is derived from one-way two-party communication complexity. Let $\mathcal{X}, \mathcal{Y}$ be input domains to a two party communication complexity problem. Alice is given $x \in \mathcal{X}$ and

Bob is given $y \in \mathcal{Y}$. Their goal is to solve some relational problem $Q \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$, where for each $(x, y) \in \mathcal{X} \times \mathcal{Y}$ the set $Q_{xy} = \{z | (x, y, z) \in Q\}$ represents the set of *correct* solutions to the communication problem.

In the one-way *communication protocol* $\mathcal{P}$, Alice must send a single message M to Bob (depending on her input X), from which Bob must output an answer in $o \in \mathcal{O}$ depending on his input Y and the message M. The maximum possible length (in bits) of M over all inputs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is the *communication cost* of the protocol $\mathcal{P}$. Communication protocols are allowed to be randomized, where each player has *private* access to an unlimited supply of random bits. The protocol $\mathcal{P}$ is said to *solve* the communication problem Q if Bob's output o belongs to Q_{xy} with failure probability at most $\delta < 1/2$. The *one-way communication complexity* of Q, denoted $R^{\rightarrow}_{\delta}(Q)$, is the minimum communication cost of a protocol which solves the protocol Q with failure probability δ.

Now a similar measure of complexity is the *distributional complexity* $D^{\rightarrow}_{\mu,\delta}(Q)$, where μ is a distribution over $\mathcal{X} \times \mathcal{Y}$, which denotes the minimum communication cost of the best deterministic protocol of Q with failure probability at most δ when the inputs $(x, y) \sim \mu$. By Yao's lemma, we have that $R^{\rightarrow}_{\delta}(Q) = \max_{\mu} D^{\rightarrow}_{\mu,\delta}(Q)$. We first review some basic facts about entropy and mutual information (see Chapter 2 of [CT12] for proofs of these facts). Recall that for a discrete random variable X supported on a finite domain Ω, the entropy $H(X)$ of X is given by $H(X) = -\sum_{a \in \Omega} \mathbf{Pr}[X = a] \log(\mathbf{Pr}[X = a])$.

PROPOSITION 4.

1. *Entropy span: If X takes on at most s values, then $0 \leq H(X) \leq \log s$.*
2. *$I(X : Y) := H(X) - H(X|Y) \geq 0$, that is, $H(X|Y) \leq H(X)$.*
3. *Chain rule: $I(X_1, X_2, \ldots, X_n : Y|Z) = \sum_{i=1}^{n} I(X_i : Y|X_1, \ldots, X_{i-1}, Z)$.*
4. *Subadditivity: $H(X, Y|Z) \leq H(X|Z) + H(Y|Z)$ and equality holds if and only if X and Y are independent conditioned on Z.*
5. *Fano's inequality: Let M be a predictor of X. In other words, there exists a function g such that $\mathbf{Pr}[g(M) = X] > 1 - \delta$, where $\delta < 1/2$. Let $\mathcal{U}$ denote the support of X, where $\mathcal{U} \geq 2$. Then $H(X|M) \leq \delta \log(|\mathcal{U}| - 1) + h_2(\delta)$, where $h_2(\delta) := \delta \log(\delta^{-1}) + (1 - \delta) \log(\frac{1}{1-\delta})$ is the binary entropy function.*

We now define the information cost of a protocol $\mathcal{P}$.

DEFINITION 7. *Let μ be a distribution of the input domain $\mathcal{X} \times \mathcal{Y}$ to a communication problem Q. Suppose the inputs (X, Y) are chosen according to μ, and let M be Alice's message to Bob, interpreted as a random variable which is a function of X and Alice's private coins. Then the information cost of a protocol $\mathcal{P}$ for Q is defined as $I(X : M)$.*

The one-way information complexity of Q with respect to μ and δ, denoted by $IC^{\rightarrow}_{\mu,\delta}(Q)$, is the minimum information cost of a one-way protocol under μ that solves Q with failure probability at most δ.

Note that by Proposition 4, we have

$$I(X : M) = H(M) - H(M|X) \leq H(M) \leq |M|,$$

where $|M|$ is the length of the message M in bits. This results in the following proposition.

PROPOSITION 5. *For every probability distribution μ on inputs,*

$$R^{\rightarrow}_{\delta}(Q) \geq IC^{\rightarrow}_{\mu,\delta}(Q).$$

7.1. Augmented indexing on large domains. We now introduce the following communication problem, known as augmented index problem on large domains. Our communication problem is derived from the communication problem (of the same name) introduced in [JW13], but we modify the guarantee of the output required so that constant probability of error is allowed. The problem is as follows.

DEFINITION 8. *Let $\mathcal{U}$ be an alphabet with $|\mathcal{U}| = k \geq 2$. Alice is given a string $x \in \mathcal{U}^d$, and Bob is given $i \in [d]$ along with the values $x_{i+1}, x_{i+2}, \ldots, x_d$. Alice must send a message M to Bob, and then Bob must output the value $x_i \in \mathcal{U}$ with probability $3/4$. We refer to this problem as the augmented index problem on large domains, and denote it by $\text{IND}_{\mathcal{U}}^d$.*

Note that in [JW13], a correct protocol is only required to determine whether $x_i = a$ for some fixed input $a \in \mathcal{U}$ given only to Bob, but such a protocol must succeed with probability $1 - \delta$. For the purposes of both problems, it is taken that $|\mathcal{U}| = \Theta(1/\delta)$. In this scenario, we note that the guarantee of our communication problem is strictly weaker, since if one had a protocol that determined whether $x_i = a$ for a given $a \in \mathcal{U}$ with probability $1 - \delta$, one could run it on all $a \in \mathcal{U}$ and union bound over all $|\mathcal{U}|$ trails, from which the exact value of x_i could be determined with probability $3/4$, thereby solving the form of the communication problem we have described. We show, nevertheless, that the same lower bound on the communication cost of our protocol holds as the lower bound in [JW13].

Let $\mathcal{X}$ be the set of all $x \in \mathcal{U}^d$, let $\mathcal{Y} = [d]$, and define μ to be the uniform distribution over $\mathcal{X} \times \mathcal{Y}$.

LEMMA 10. *Suppose $|\mathcal{U}| \geq c$ for some sufficiently large constant c. We have $IC^{\rightarrow}_{\mu,3/4}(\text{IND}_{\mathcal{U}}^d) \geq d \log(|\mathcal{U}|)/2$.*

Proof. Fix any protocol $\mathcal{P}$ for $\text{IND}_{\mathcal{U}}^d$ which fails with probability at most $1/4$. Let $X = (X_1, X_2, \ldots, X_d)$ denote Alice's input as chosen via μ, and let M be Alice's message to Bob given X. By Proposition 4

$$\begin{aligned} I(X : M) &= \sum_{i=1}^{d} I(X_i : M | X_1, \ldots, X_{i-1}) \\ &= \sum_{i=1}^{d} \Big(H(X_i | X_1, \ldots, X_{i-1}) - H(X_i | M, X_1, \ldots, X_{i-1}) \Big). \end{aligned}$$

First note that since X_i is independent of X_j for all $j \neq i$, we have $H(X_i | X_1, \ldots, X_{i-1}) = H(X_i) = \log(|\mathcal{U}|)$. Now since the protocol $\mathcal{P}$ is correct on $\text{IND}_{\mathcal{U}}^d$, then the variables $M, X_1, \ldots, X_{i-1}$ must be a predictor for X_i with failure probability $1/4$ (since Bob outputs X_i with probability $3/4$ given only $M, X_1, \ldots, X_{i-1}$ and his private, independent randomness). So by Fano's inequality (Proposition 4), we have

$$\begin{aligned} H(X_i | M, X_1, \ldots, X_{i-1}) &\leq \frac{1}{4} \log(|\mathcal{U}| - 1) + h_2\left(\frac{1}{4}\right) \\ &\leq \frac{1}{2} \log(|\mathcal{U}|), \end{aligned}$$

which holds when $|\mathcal{U}|$ is sufficiently large. Putting this together, we obtain

$$I(X : M) \geq \frac{d \log(|\mathcal{U}|)}{2}.$$

∎

COROLLARY 5. *We have* $R^{\rightarrow}_{3/4}(\text{IND}^d_{\mathcal{U}}) = \Omega(d\log(|\mathcal{U}|))$.

We now use this lower bound on $\text{IND}^d_{\mathcal{U}}$ to show that, even when the index output is from a distribution with constant *additive* error from the true L_p distribution, returning an estimate with probability $1-\delta_2$ still requires $\Omega(\epsilon^{-p}\log(n)\log(1/\delta_2))$ bits of space.

THEOREM 10. *Fix any $p > 0$ constant bounded away from 0, and let $\epsilon < 1/3$ with $\epsilon^{-p} = o(n)$. Then any L_p sampling algorithm that outputs* FAIL *with probability at most $1/100$, and otherwise returns an item $\ell \in [n]$ such that $\mathbf{Pr}[\ell = l] = |f_l|^p/\|f\|_p^p \pm 1/50$ for all $l \in [n]$, along with an estimate $\tilde{f}_\ell$ such that $\tilde{f}_\ell = (1\pm\epsilon)f_\ell$ with probability $1-\delta_2$, requires $\Omega(\epsilon^{-p}\log(n)\log(1/\delta_2))$ bits of space.*

Proof. We reduce via $\text{IND}^d_{\mathcal{U}}$. Suppose we have a streaming algorithm $\mathcal{A}$ which satisfies all the properties stated in the theorem. Set $|\mathcal{U}| = 1/(10\delta_2)$, and let $X \in \mathcal{U}^d$ be Alice's input, where $d = rs$, where $r = \frac{1}{10^{p+1}\epsilon^p}$ and $s = \log(n)$. Alice conceptually divides X into s blocks $X^1, \ldots, X^s$, each containing r items $X^i = X^i_1, X^i_2, \ldots, X^i_r \in \mathcal{U}$. Fix some labeling $\mathcal{U} = \{\sigma_1, \ldots, \sigma_k\}$, and let $\pi(X^i_j) \in [k]$ be such that $X^i_j = \sigma_{\pi(X^i_j)}$. Then each X^i_j can be thought of naturally as a binary vector in $\mathbb{R}^{rsk}$ with support 1, where $(X^i_j)_t = 1$ when $t = (i-1)r + (j-1)k + \pi(X^i_j)$, and $(X^i_j)_t = 0$ otherwise. Set $n' = rsk < n$ for $\epsilon^{-p} = o(n)$. Using this interpretation of $X^i_j \in \mathbb{R}^{rsk}$, we define the vector $f \in \mathbb{R}^{rsk}$ by

$$f = \sum_{i=1}^{s}\sum_{j=1}^{r} B^i X^i_j,$$

where $B = 10^{1/p}$. Alice can construct a stream with the frequency vector f by making the necessary insertions, and then send the state of the streaming algorithm $\mathcal{A}$ to Bob. Now Bob has some index $i^* \in [d] = [rs]$, and his goal is to output the value of $X^{i'}_{j'} = X_{i^*}$ such that $i^* = (i'-1)r + j'$. Since Bob knows X^j_i for all (i,j) with $i > i'$, he can delete off the corresponding values of $B^i X^i_j$ from the stream, leaving the vector f with the value

$$f = \sum_{i=1}^{i'}\sum_{j=1}^{r} B^i X^i_j.$$

For $j \in [k]$, let $\gamma^j \in \mathbb{R}^{rsk}$ be the binary vector with $\gamma^j_{(i'-1)r+(j'-1)k+j} = B^{i'}/(10\epsilon)$ and $\gamma^j_t = 0$ at all other coordinates $t \neq (i'-1)r + (j'-1)k + j$. Bob then constructs the streams $f^j = f + \gamma^j$ for $j = 1, \ldots, k$ sequentially. After he constructs f^j, he runs $\mathcal{A}$ on f^j to obtain an output $(\ell_j, \tilde{f}^j_{\ell_j}) \in ([n'] \times \mathbb{R}) \cup (\{\texttt{FAIL}\} \times \{\texttt{FAIL}\})$ from the streaming algorithm, where if the algorithm did not fail we have that $\ell_j \in [n']$ is the index output and $\tilde{f}^j_{\ell_j}$ is the estimate of $f^j_{\ell_j}$. By union bounding over the guarantee of $\mathcal{A}$ we have that if $\ell_j \neq \texttt{FAIL}$, then $\tilde{f}_{\ell_j} = (1\pm\epsilon)f^j_{\ell_j}$ for all $j = 1, 2, \ldots, k$ with probability $1 - k\delta_2 > 9/10$. Call this event $\mathcal{E}_1$. Conditioned on $\mathcal{E}_1$, it follows that if for each ℓ_j with $\ell_j = (i'-1)r + (j'-1)k + j$, if $X^{i'}_{j'} = \sigma_j$, then

$$\tilde{f}^j_{\ell_j} > B^{i'}\left(1 + \frac{1}{10\epsilon}\right)(1-\epsilon) > \frac{B^{i'}}{10\epsilon} + \frac{9}{10}B^{i'} - \epsilon B^{i'}.$$

On the other hand, if $X_{j'}^{i'} \neq \sigma_j$, then we will have

$$\tilde{f}_{\ell_j}^{j} < (B^{i'}/(10\epsilon))(1+\epsilon) = \frac{B^{i'}}{10\epsilon} + \frac{B^{i'}}{10}$$
$$< \frac{B^{i'}}{10\epsilon} + \frac{9}{10}B^{i'} - \epsilon B^{i'}$$

using that $\epsilon < 1/3$. Thus if $\ell_j = (i'-1)r + (j'-1)k + j$, Bob can correctly determine whether or not $X_{j'}^{i'} = \sigma_j$. Now suppose that, in actuality, Alice's item was $X_{j'}^{i'} = \sigma_\tau \in \mathcal{U}$ for some $\tau \in [k]$. Set $\lambda = (i'-1)r + (j'-1)k + \tau$. To complete the proof, it suffices to lower bound the probability that $\ell_\tau \neq \lambda$.

Thus we consider only the event of running $\mathcal{A}$ on f^τ. We know that with probability 99/100, $\ell_\tau \neq \texttt{FAIL}$. We write $\mathcal{E}_2$ to denote the event that $\ell_\tau \neq \texttt{FAIL}$. Let $f_{-\lambda}$ be equal to f everywhere except with the coordinate λ set equal to 0. Then

$$\|f_{-\lambda}^\tau\|_p^p < \sum_{i=1}^{i'}\sum_{j=1}^{r}(B^p)^i$$
$$\leq r\sum_{i=1}^{i'} 10^i \leq \left(\frac{1}{10^{p+1}\epsilon^p}\right)\frac{10^{i'+1}}{9},$$

so

$$\frac{|f_\lambda^\tau|^p}{\|f_{-\lambda}^\tau\|_p^p} \geq \frac{10^{i'}(\frac{1}{10\epsilon})^p}{(\frac{1}{10^{p+1}\epsilon^p})\frac{10^{i'+1}}{9}} \geq \frac{9(\frac{1}{10\epsilon})^p}{(\frac{1}{10\epsilon})^p} \geq 9.$$

Since $\mathcal{A}$ has 1/50-additive error, we conclude $\mathbf{Pr}[\ell_\tau = \lambda] > 9/10 - 1/50 = 22/25$, and call the event that this occurs $\mathcal{E}_3$. Then conditioned on $\mathcal{E} = \mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3$ Bob sucsessfully recovers the value of $X_{j'}^{i'} = X_{i^*}$, and thus solves the communication problem. Note that the probability of success is $\mathbf{Pr}[\mathcal{E}] > 1 - (1/10 + 1/100 + 3/25) > 3/4$, and thus this protocol solves $\text{IND}_{\mathcal{U}}^d$. So by Corollary 5, it follows that any such streaming algorithm $\mathcal{A}$ requires $\Omega(rs\log(|\mathcal{U}|)) = \Omega(\epsilon^{-p}\log(n)\log(1/\delta_2))$ bits of space. Note that the stream f in question had length $n' < n$ for p constant bounded from 0, and no coordinate in the stream ever had a value greater than poly(n), and thus the stream in question is valid in the given streaming model. ∎

8. Conclusion. This work demonstrates the existence of perfect L_p samplers for $p \in (0,2)$ using $O(\log^2(n)\log(1/\delta))$ bits of space in the random oracle model. This bound is tight in terms of both n and δ. However, to derandomize our algorithm for $p < 2$, our space increases by a $O((\log\log n)^2)$-factor, which is perhaps unnecessary. There are also several other open problems for L_p samplers which this work does not close. Notably, there is still a $\log(n)$ factor gap between the upper and lower bounds for L_2 samplers, as the best known lower bound for any $p \geq 0$ is $\Omega(\log^2 n)$, compared to our upper bound of $O(\log^3 n)$. While perfect L_2 samplers using polylogarithmic space were not known before this work, our upper bound matches the best upper bounds of prior approximate L_2 samplers with constant $\nu = \Omega(1)$. It is therefore an open question whether this additional factor of $\log n$ is required in the space complexity of an L_2 sampler, perfect or otherwise.

Second, one notable shortcoming of the perfect sampler presented in this paper is the large update time. To obtain a perfect sampler as defined in the introduction, the algorithm in this paper takes polynomial (in n) time to update its data structures after each entry in the stream. This is clearly nonideal, since most streaming applications

demand constant or polylogarithmic update time. Using our rounding procedure, we can obtain a $(1 \pm 1/\mathrm{poly}(\log n))$ relative error sampler with polylogarithmic update time (and the same space as the perfect sampler), but it is still an open problem to design a perfect L_p sampler with optimal space dependency as well as polylogarithmic update time.

Finally, there are several gaps in the dependency on ϵ, δ_2 in our procedure which, in addition to outputting an index $i \in [n]$, also outputs a $(1 \pm \epsilon)$ estimate of the frequency f_i. Taking Theorem 10 along with the known lower bounds for L_p sampling, our best lower bound for the problem is $\Omega(\log^2(n)\log(1/\delta_1) + \epsilon^{-p}\log(n)\log(1/\delta_2))$, where δ_1 is the probability that the sampler fails to output an index i. On the other hand, our best upper bound is $O\big(\big(\log^2(n)(\log\log n)^2 + \beta\log(n)\log(1/\delta_2)\big)\log(1/\delta_1)\big)$ for $p \in (0,2)$, and $O\big(\big(\log^3(n) + \epsilon^{-2}\log^2(n)\log(1/\delta_2)\big)\log(1/\delta_1)\big)$ for $p = 2$, where $\beta = \min\big\{\epsilon^{-2}, \epsilon^{-p}\log\big(\frac{1}{\delta_2}\big)\big\}$. Notably, the $\log(1/\delta_1)$ multiplies the $\log(1/\delta_2)$ term in the upper bound but not in the lower bound. We leave it as an open problem to determine precisely the right dependencies of such an algorithm on $\epsilon, \delta_1, \delta_2$.

Appendix A. Original L_p sampling via count-sketch. In a previous version of this work, we used a slightly different testing algorithm for the L_p sampler. Namely, we used the classic count-sketch estimation procedure of Theorem 1 to obtain a y such that $\|y-\zeta\|_\infty$ is small. We then take the largest coordinate of y as our guess of the maximizer in ζ. The algorithm presented in the current version has the advantage of being slightly simpler and does not incur the $(\log\log n)^2$ blow-up in space for $p = 2$ from the derandomization. In this section, we show how the algorithm in the original version can be derandomized using the general derandomization results for linear sketches of Theorem 5. First, we introduce a few preliminary tools that we will need.

A.1. Preliminaries. We first introduce the L_2 estimation algorithm of [Ind06]. To estimate $\|f\|_2$ for $f \in \mathbb{R}^n$, we generate i.i.d. Gaussians $\varphi_{i,j} \sim \mathcal{N}(0,1)$ for $i \in [n]$ and $j \in [r]$, where $r = \Theta(\log(n))$. We will later derandomize this assumption. We then store the vector $B \in \mathbb{R}^r$, where $B_j = \sum_{i=1}^n f_i\varphi_{i,j}$ for $j = 1, \ldots, r$, which can be computed update by update throughout the stream. We return the estimate $R = \mathrm{median}_j \frac{5|B_j|}{4}$.

LEMMA 11. *For any constant $c > 0$, the value of R as computed in the above algorithm satisfies $\frac{1}{2}\|f\|_2 \le R \le 2\|f\|_2$ with probability $1 - n^{-c}$.*

Proof. Each coordinate B_j is distributed as $|B_j| = |g_j|\|f\|_2$, where g_j are i.i.d. Gaussian random variables. A simple computation shows that $\mathbf{Pr}[|g_j| \in [2/5, 8/5]] > .55$, and thus $\mathbf{Pr}[(5/4)|B_j| \in [1/2\|f\|_2, 2\|f\|_2]] > .55$. Then by Chernoff–Hoeffding bounds, the median of $O(\log(n))$ repetitions satisfies this bound with probability $1 - n^{-c}$ as stated. ∎

Finally, we remark that making a simple modification to the classic count-sketch algorithm (see Theorem 1) still results in the same error guarantee. Let $A \in \mathbb{R}^{d\times k}$ be a $d\times k$ count-sketch matrix. The modification is as follows: instead of each variable $h_i(\ell)$ being uniformly distributed in $\{1, 2, \ldots, k\}$, we replace them with variables $h_{i,j,\ell} \in \{0,1\}$ for $(i,j,\ell) \in [d]\times[k]\times[n]$, such that $h_{i,j,\ell}$ are all i.i.d. and equal to 1 with probability $1/k$. We also let $g_{i,h,\ell} \in \{1,-1\}$ be i.i.d. Rademacher variables (1 with probability 1/2). Then $A_{i,j} = \sum_{\ell=1}^n f_\ell g_{i,j,\ell}h_{i,j,\ell}$, and the estimate y_ℓ of f_ℓ is given by

$$y_\ell = \mathrm{median}\{g_{i,j,\ell}A_{i,j} \mid h_{i,j,\ell} = 1\}.$$

Thus the element f_ℓ can be hashed into multiple buckets in the same row of A, or even be hashed into none of the buckets in a given row. By Chernoff bounds,

L_p `Sampler`

1. For $0 < p < 2$, set $\epsilon = \Theta(1)$, and for $p = 2$, set $\epsilon = \Theta(\sqrt{1/\log(n)})$. Let $d = \Theta(\log(n))$, and instantiate a $d \times 6/\epsilon^2$ count-sketch table A, and set $\mu \sim \text{Uniform}[\frac{1}{2}, \frac{3}{2}]$.
2. Duplicate updates to f to obtain the vector $F \in \mathbb{R}^{n^c}$ so that $f_i = F_{i_j}$ for all $i \in [n]$ and $j = 1, 2, \ldots, n^{c-1}$ for some fixed constant c.
3. Choose i.i.d. exponential random variables $t = (t_1, t_2, \ldots, t_{n^c})$, and construct the stream $\zeta_i = F_i \cdot \mathtt{rnd}_\nu(1/t_i^{1/p})$.
4. Run A on ζ to obtain an estimate y with $\|y - |\zeta|\|_\infty < \epsilon\|\zeta_{\text{tail}(1/\epsilon^2)}\|_2$ as in Theorem 11.
5. Run L_2 estimator on ζ to obtain $R \in [\frac{1}{2}\|\zeta\|_2, 2\|\zeta\|_2]$ with high probability.
6. If $y_{(1)} - y_{(2)} < 100\mu\epsilon R$ or if $y_{(2)} < 50\epsilon\mu R$, report `FAIL`, else return $i \in [n]$ such that $y_{i_j} = y_{(1)}$ for some $j \in [n^{c-1}]$.

FIG. 6. *Our main L_p sampling algorithm.*

$|\{g_{i,j,\ell}A_{i,j} \mid h_{i,j,\ell} = 1\}| = \Theta(d)$ with high probability for all $\ell \in [n]$. Observe that the marginal distribution of each bucket is the same as before, and thus the original analysis of count-sketch ([CCFC02a]) is unchanged, as it only relies on taking the median of $\Theta(d)$ buckets, each of which independently succeed in giving a good estimate with probability at least $2/3$, as is the case here. Thus the bounds of Theorem 1 apply as usual.

THEOREM 11. *Let $A \in \mathbb{R}^{d\times k}$ be the modified count-sketch as described above. If $d = \Theta(\log(n))$, $k = 6/\epsilon^2$, and $c \geq 1$ is any constant, then we have $\|y - f\|_\infty < \epsilon\|f_{tail(1/\epsilon^2)}\|_2$ with probability $1 - n^{-c}$.*

A.2. The L_p sampler. We begin by describing the original sampling algorithm, as shown in Figure 6. The algorithm duplicates coordinates just as the sampler of Figure 3 and scales it by inverse $1/p$th powers of i.i.d. exponentials $1/t_i^{1/p}$. We also perform the same rounding procedure, turning z into ζ. Having constructed the transformed stream ζ, we then run a $\Theta(\log(n)) \times 6/\epsilon^2$ instance A of count-sketch on ζ to obtain an estimate vector y with $\|y - |\zeta|\|_\infty < \epsilon\|\zeta_{\text{tail}(1/\epsilon^2)}\|_2$ with probability $1 - n^{-c}$ (as in Theorem 11). Here, for a vector $v \in \mathbb{R}^n$, $|v| \in \mathbb{R}^n$ is the vector such that $(|v|)_i = |v_i|$ for all $i \in [n]$. Thus y_j is an estimate of the *absolute value* ζ_j and is always positive. This is simply accomplished by taking the absolute value of the usual estimate y obtained from count-sketch.

Then for $0 < p < 2$, we set $\epsilon = \Theta(1)$, and for $p = 2$, we set $\epsilon = \Theta(1/\sqrt{\log(n)})$. Next, we obtain estimates $R \in [\frac{1}{2}\|\zeta\|_2, 2\|\zeta\|_2]$ via the algorithm of Lemma 11 with high probability. The algorithm then finds $y_{(1)}, y_{(2)}$ (the two largest coordinates of y) and samples $\mu \sim \text{Uniform}[1/2, 3/2]$. It then checks if $y_{(1)} - y_{(2)} < 100\mu\epsilon R$ or if $y_{(2)} < 50\epsilon\mu R$ and reports `FAIL` if either occur; otherwise it returns $i \in [n]$ with $y_{i_j} = y_{(1)}$ for some $j \in [n^{c-1}]$.

Let $i^* \in [n^c]$ be the index of the maximizer in y, so $y_{i^*} = y_{(1)}$. By checking that $y_{(1)} - y_{(2)} > 100\mu\epsilon R$ and noting that $100\mu\epsilon R \geq 25\|y - |\zeta|\|_\infty$ and $z_k = (1 \pm \nu)\zeta_k$ for all $k \in [n^c]$, for $\nu < \epsilon$ sufficiently small we ensure that $|z_{i^*}|$ is also the maximum element in z. The necessity for the test $y_{(2)} \geq 50\epsilon\mu R$ is less straightforward (see Remark 2 for justification). To prove correctness, we need to analyze the conditional probability of failure given $D(1) = i$. Let $N = |\{i \in [n^c] \mid F_i \neq 0\}|$ (N is the support size of F). We

can assume that $N \neq 0$ (to check this, one could run, for instance, the $O(\log^2(n))$-bit support sampler of [JST11]). Note that $n^{c-1} \leq N \leq n^c$. We now will prove the propositions and lemmas needed to demosntrate correctness of this sampler. Lemmas 12 and 13 are the analogous results to Lemmas 3 and 4 in section 4 and will follow nearly the same proofs.

PROPOSITION 6. *Let $X, Y \in \mathbb{R}^d$ be random variables where $Z = X + Y$. Suppose X is independent of some event E, and let $M > 0$ be such that for every $i \in [d]$ and every $a < b$ we have $\mathbf{Pr}[a \leq X_i \leq b] \leq M(b-a)$. Suppose further that $|Y|_\infty \leq \epsilon$. Then if $I = I_1 \times I_2 \times \cdots \times I_d \subset \mathbb{R}^n$, where each $I_j = [a_j, b_j] \subset \mathbb{R}$, $-\infty \leq a_j < b_j \leq \infty$ is a (possibly unbounded) interval, then*

$$\mathbf{Pr}[Z \in I|E] = \mathbf{Pr}[Z \in I] + O(\epsilon d M).$$

Proof. For $j \in [d]$, let $\overline{I}_j = [a_j - \epsilon, b_j + \epsilon], \underline{I}_j = [a_j + \epsilon, b_j - \epsilon]$, and let $\overline{I} = \overline{I}_1 \times \cdots \times \overline{I}_d$, and $\underline{I} = \underline{I}_1 \times \cdots \times \underline{I}_d$. If one of the endpoints is unbounded we simply use the convention $\infty \pm c = \infty$, $-\infty \pm c = -\infty$ for any real c. Then

$$\mathbf{Pr}[Z \in I|E] \leq \mathbf{Pr}[X \in \overline{I}|E] = \mathbf{Pr}[X \in \overline{I}]$$

$$\leq \mathbf{Pr}[X \in \underline{I}] + \mathbf{Pr}\left[\bigcup_{i=1}^{d} X_i \in \overline{I}_j \setminus \underline{I}_j\right].$$

By the union bound, this is at most $\mathbf{Pr}[X \in \underline{I}] + 4d\epsilon M \leq \mathbf{Pr}[Z \in I] + 4d\epsilon M$. Similarly, $\mathbf{Pr}[Z \in I|E] \geq \mathbf{Pr}[X \in \underline{I}] \geq \mathbf{Pr}[X \in \overline{I}] - 4d\epsilon M \geq \mathbf{Pr}[Z \in I] - 4d\epsilon M$. ∎

LEMMA 12. *For $p \in (0, 2]$ a constant bounded away from 0 and any $\nu \geq n^{-c}$, $\mathbf{Pr}[\neg\texttt{FAIL} \mid D(1)] = \mathbf{Pr}[\neg\texttt{FAIL}] \pm O(\log(n)\nu)$ for every possible $D(1) \in [N]$.*

Proof. By Lemma 2, conditioned on $\mathcal{E}_1$, for every $k < N - n^{9c/10}$ we have $|z_{D(k)}| = U_{D(k)}^{1/p}(1 \pm O(n^{-c/10}))^{1/p} = U_{D(k)}^{1/p}(1 \pm O(\frac{1}{p}n^{-c/10}))$ (using the identity $(1+x) \leq e^x$ and the Taylor expansion of e^x), where $U_{D(k)} = (\sum_{\tau=1}^{k} \frac{E_\tau}{\mathbb{E}[\sum_{j=\tau}^{N} |F_{D(j)}|^p]})^{-1}$ is independent of the antirank vector D (in fact, it is totally determined by k and the hidden exponentials E_i). Then for c sufficiently large, we have $|\zeta_{D(k)}| = U_{D(k)}^{1/p}(1 \pm O(\nu))$, and so for all $p \in (0, 2]$ and $k < N - n^{9c/10}$

$$|\zeta_{D(k)}| = U_{D(k)}^{1/p} + U_{D(k)}^{1/p} V_{D(k)},$$

where $V_{D(k)}$ is some random variable that satisfies $|V_{D(k)}| = O(\nu)$. Now consider a bucket $A_{i,j}$ for $(i,j) \in [d] \times [6/\epsilon^2]$. Let $\sigma_k = \texttt{sign}(z_k) = \texttt{sign}(\zeta_k)$ for $k \in [n^c]$. Then we write $A_{i,j} = \sum_{k \in B_{ij}} \sigma_{D(k)} |\zeta_{D(k)}| g_{i,j,D(k)} + \sum_{k \in S_{ij}} \sigma_{D(k)} |\zeta_{D(k)}| g_{i,j,D(k)}$, where $B_{ij} = \{k \leq N - n^{9c/10} \mid h_{i,j,D(k)} = 1\}$ and $S_{ij} = \{n^c \geq k > N - n^{9c/10} \mid h_{i,j,D(k)} = 1\}$ (see the notation above Theorem 11). Here we define $\{D(N+1), \ldots, D(n^c)\}$ to be the set of indices i with $F_i = 0$ (in any ordering, as they contribute nothing to the sum). So

$$A_{i,j} = \sum_{k \in B_{ij}} g_{i,j,D(k)} \sigma_{D(k)} U_{D(k)}^{1/p} + \sum_{k \in B_{ij}} g_{i,j,D(k)} \sigma_{D(k)} U_{D(k)}^{1/p} V_{D(k)} + \sum_{k \in S_{ij}} g_{i,j,D(k)} \zeta_{D(k)}.$$

Importantly, observe that since the variables $h_{i,j,D(k)}$ are fully independent, the sets $B_{i,j}, S_{i,j}$ are independent of the antirank vector D. In other words, the values $h_{i,j,D(k)}$

are independent of the values $D(k)$ (and of the entire antirank vector). Note that this would not necessarily be the case if these were only ℓ-wise independent for some $\ell = o(n^c)$. So we can condition on a fixed set of values $\{h_{i,j,D(1)}, \ldots, h_{i,j,D(n^c)}\}$ now, which fixes the sets $B_{i,j}, S_{i,j}$.

CLAIM 2. *For all i, j, and $p \in (0, 2]$, we have $|\sum_{k\in B_{ij}} g_{i,j,D(k)}\sigma_{D(k)}U_{D(k)}^{1/p}V_{D(k)}| + |\sum_{k\in S_{ij}} g_{i,j,D(k)}\zeta_{D(k)}| = O(\sqrt{\log(n)}\nu\|z\|_2)$ with probability $1 - O(\log^2(n)n^{-c})$.*

Proof. By Khintchines's inequality (Fact 1), we have $|\sum_{k\in S_{ij}} g_{i,j,D(k)}\zeta_{D(k)}| = O(\sqrt{\log(n)})(\sum_{k\in S_{i,j}}(2z_{D(k)})^2)^{1/2}$ with probability $1 - n^{-c}$. This is a sum over a subset of the $n^{9c/10}$ smallest items $|z_i|$, and thus $\sum_{k\in S_{i,j}} z_{D(k)}^2 < \frac{n^{9c/10}}{N}\|z\|_2^2$, giving $|\sum_{k\in S_{ij}} g_{i,j,D(k)}\zeta_{D(k)}| = O(\sqrt{\log(n)}n^{-c/30}\|z\|_2)$. Furthermore, using the fact that for $k \le N - n^{9c/10}$ we have $|\zeta_{D(k)}| < 2U_{D(k)}^{1/p}$ and $|V_{D(k)}| = O(\nu)$, we have $|\sum_{k\in B_{ij}} g_{i,j,D(k)}\sigma_{D(k)}U_{D(k)}^{1/p}V_{D(k)}| = O(\sqrt{\log(n)}\nu\|z\|_2)$ with probability $1-n^{-c}$ again by Khintchine's inequality, as needed. Note that there are only $O(\epsilon^{-2}\log(n)) = O(\log^2(n))$ (for $p < 2$ this is $O(\log(n))$) terms

$$\left|\sum_{k\in B_{ij}} g_{i,j,D(k)}\sigma_k U_{D(k)}^{1/p}V_{D(k)}\right| + \left|\sum_{k\in S_{ij}} g_{i,j,D(k)}\zeta_{D(k)}\right|$$

which ever occur in all of the $A_{i,j}$'s, since the count-sketch has size $O(\epsilon^{-2}\log(n))$. Union bounding over these buckets, and taking c sufficiently large, the claim follows. ∎

Call the event where Claim 2 holds $\mathcal{E}_2$. Conditioned on $\mathcal{E}_2$, we can decompose $|A_{i,j}|$ for all i, j into $|\sum_{k\in B_{ij}} g_{i,j,D(k)}\sigma_{D(k)}U_{D(k)}^{1/p}| + \mathcal{V}_{ij}$, where $\mathcal{V}_{ij}$ is some random variable satisfying $|\mathcal{V}_{ij}| = O(\sqrt{\log(n)}\nu\|z\|_2)$ and $\sum_{k\in B_{ij}} g_{i,j,D(k)}\sigma_{D(k)}U_{D(k)}^{1/p}$ is independent of the antirank vector D (it depends only on the hidden exponentials E_k and the uniformly random signs $g_{i,j,D(k)}\sigma_{D(k)}$). Let $U_{ij}^* = |\sum_{k\in B_{ij}} g_{i,j,D(k)}\sigma_{D(k)}U_{D(k)}^{1/p}|$. Let $\Gamma(k) = \{(i,j) \in [d] \times [k] \mid h_{i,j,D(k)} = 1\}$. Then our estimate for $|\zeta_{D(k)}|$ is $y_{D(k)} = \text{median}_{(i,j)\in\Gamma(k)}\{U_{i,j}^* + \mathcal{V}_{i,j}\} = \text{median}_{(i,j)\in\Gamma(k)}\{U_{i,j}^*\} + \mathcal{V}_{D(k)}^*$, where $|\mathcal{V}_{D(k)}^*| = O(\sqrt{\log(n)}\nu\|z\|_2)$ for all $k \in [n^c]$.

We now consider our L_2 estimate, which is given by $R = \frac{5}{4}\text{median}_j\{|\sum_{k\in[n^c]}\varphi_{kj}\zeta_k|\}$, where the φ_{kj}'s are i.i.d. normal Gaussians. We can write this as

$$R = \frac{5}{4}\text{median}_j\left\{\left|\sum_{k\in B}\varphi_{D(k)j}\sigma_{D(k)}U_{D(k)}^{1/p} + \left(\sum_{k\in B}\varphi_{D(k)j}\sigma_{D(k)}U_{D(k)}^{1/p}V_{D(k)} + \sum_{k\in S}\varphi_{D(k)j}\zeta_{D(k)}\right)\right|\right\},$$

where $B = \cup_{ij}B_{ij}$ and $S = [n^c] \setminus B$. Now the $\varphi_{D(k)j}$'s are not ± 1 random variables, so we cannot apply Khintchine's inequality. However, by the 2-stability of Gaussians (Definition 2), if $\varphi_1, \ldots, \varphi_n$ are i.i.d. Gaussian, then $\mathbf{Pr}[|\sum_i \varphi_i a_i| > O(\sqrt{\log(n)})\|a\|_2]$ $= \mathbf{Pr}[|\varphi|\|a\|_2 > O(\sqrt{\log(n)})\|a\|_2]$, where φ is again Gaussian. This latter probability can be bounded by n^{-c} via the pdf of a Gaussian, which is the same bound as Khintchine's inequality. So applying the same argument as in Claim 2, we have $R = \frac{5}{4}\text{median}_j\{(|\sum_{k\in B}\varphi_{D(k)j}\sigma_{D(k)}U_{D(k)}^{1/p}|\} + \mathcal{V}_R$ with probability $1 - O(n^{-c})$, where $|\mathcal{V}_R| = O(\sqrt{\log(n)}\nu\|z\|_2)$. Call this event $\mathcal{E}_3$. By the symmetry of Gaussians, the

value $\varphi_{D(k)j}\sigma_{D(k)}$ is just another i.i.d. Gaussian, so $|\sum_{k\in B}\varphi_{D(k)j}\sigma_{D(k)}U^{1/p}_{D(k)}|$ is independent of the antirank vector.

Let $U^*_{D(k)} = \text{median}_{(i,j)\in\Gamma(k)}\{U^*_{i,h_i(D(k))}\}$ for $k \in [n^c]$, and

$$U^*_R = \frac{5}{4}\text{median}_j\left(\left|\sum_{k\in B}\varphi_{D(k)j}\sigma_{D(k)}U^{1/p}_{D(k)}\right|\right).$$

Then both $U^*_{D(k)}, U^*_R$ are independent of the antiranks $D(k)$ (the former does, however, depend on k), and $y_{D(k)} = U^*_{D(k)} + V^*_{D(k)}$. Now to analyze our failure condition, we define a deterministic function $\Lambda(x,v) \in \mathbb{R}^2$. For vector x and a scalar v, set $\Lambda(x,v)_1 = x_{(1)} - x_{(2)} - 100\epsilon v$ and $\Lambda(x,v)_2 = x_{(2)} - 50\epsilon v$. Note $\Lambda(y,\mu R) \geq 0$ (coordinatewise) if and only if $\neg$FAIL.

Claim 3. *Conditioned on $\mathcal{E}_1\cap\mathcal{E}_2\cap\mathcal{E}_3$, we have the decomposition $\Lambda(y,\mu R) = \Lambda(\vec{U}^*,\mu U^*_R)+\overline{V}$, where the former term is independent of the max index and $\|\overline{V}\|_\infty = O(\sqrt{\log(n)}\nu\|z\|_2)$.*

Proof. We have shown that $|\mathcal{V}_\mathcal{R}|$ and $|\mathcal{V}^*_{D(k)}|$ are both $O(\sqrt{\log(n)}\nu\|z\|_2)$ for all $k \in [n^c]$ conditioned on $\mathcal{E}_1\cap\mathcal{E}_2\cap\mathcal{E}_3$. We have $y = \vec{U}^* + \vec{\mathcal{V}}^*$, where $\vec{U}^*_{D(k)} = U^*_{D(k)}$ and $\vec{V}^*_{D(k)} = V^*_{D(k)}$, so $\vec{\mathcal{V}}^*$ can change the value of the two largest coordinates in y by at most $\|\vec{\mathcal{V}}^*\|_\infty = O(\sqrt{\log(n)}\nu\|z\|_2)$. Similarly $|\mathcal{V}_\mathcal{R}|$ can change the value of R by at most $O(\sqrt{\log(n)}\nu\|z\|_2)$, which completes the proof of the decomposition. To see the claim of independence, note that $\Lambda(\vec{U}^*,\mu U^*_R)$ is a deterministic function of the hidden exponentials $E_1,\dots,E_N$, the random signs g, and the uniform random variable μ, the joint distribution of all of which is marginally independent of the antirank vector D, which completes the claim. ∎

To complete the proof of the lemma, it suffices to show the anticoncentration of $\Lambda(\vec{U}^*,\mu U^*_R)$. Now for any interval I

$$\begin{aligned}\mathbf{Pr}[\Lambda(\vec{U}^*,\mu U^*_R)_1 \in I] &= \mathbf{Pr}[\mu \in I'/(100\epsilon U^*_R)]\\ &= O(|I|/(\epsilon U^*_R))\end{aligned}$$

and

$$\begin{aligned}\mathbf{Pr}[\Lambda(\vec{U}^*,\mu U^*_R)_2 \in I] &= \mathbf{Pr}[\mu \in I''/(50\epsilon U^*_R)]\\ &= O(|I|/(\epsilon U^*_R)),\end{aligned}$$

where I' and I'' are the result of shifting the interval I by a term which is independent of μ. Here $|I| \in [0,\infty]$ denotes the size of the interval I. Thus it suffices to lower bound U^*_R. We have $2U^*_R > R > \frac{1}{2}\|z\|_2$ after conditioning on the success of our L_2 estimator, an event we call $\mathcal{E}_4$, which holds with probability $1-n^{-c}$ by Lemma 11. Thus $\mathbf{Pr}[\Lambda(\vec{U}^*,\mu U^*_R)_1 \in I] = O(\epsilon^{-1}|I|/\|z\|_2)$ and $\mathbf{Pr}[\Lambda(\vec{U}^*,\mu U^*_R)_2 \in I] = O(\epsilon^{-1}|I|/\|z\|_2)$ for any interval I. So by Proposition 6, conditioned on $\mathcal{E}_1\cap\mathcal{E}_2\cap\mathcal{E}_3\cap\mathcal{E}_4$ we have

$$\mathbf{Pr}\Big[\Lambda(y,\mu R) \geq \vec{0} \in \mathbb{R}^2 \;\Big|\; D(1)\Big] = \mathbf{Pr}\Big[\Lambda(y,\mu R) \geq \vec{0}\Big] \pm O(\log(n)\nu). \tag{6}$$

Note that $\mathcal{E}_1\cap\mathcal{E}_2\cap\mathcal{E}_3\cap\mathcal{E}_4$ holds with probability $1-O(n^{-c+1})$, so choosing c such that $n^{-c} < \log(n)\nu$, equation (6) holds without conditioning on $\mathcal{E}_1\cap\mathcal{E}_2\cap\mathcal{E}_3\cap\mathcal{E}_4$, which completes the proof of the lemma. ∎

LEMMA 13. *If y is the vector obtained via count-sketch as in the algorithm L_p Sampler, and $0 < p \leq 2$ a constant, then we have $\mathbf{Pr}[y_{(1)} - y_{(2)} > 100\epsilon\mu R,\ y_{(2)} > 50\epsilon\mu R] \geq 1/2$, where $\epsilon = \Theta(1)$ when $p < 2$, and $\epsilon = \Theta(1/\sqrt{\log(n)})$ when $p = 2$.*

Proof. By Proposition 1, with probability $1 - 3e^{-4} > .9$ we have $\|z_{tail(16)}\|_2 = O(|F\|_p)$ for $p < 2$, and $\|z_{tail(16)}\| = O(\sqrt{\log(n)}\|F\|_p)$ when $p = 2$. Observe that for $t \in [16]$ we have $|z_{D(t)}| < \|F\|_p(\frac{2}{\sum_{\tau=1}^{t} E_\tau})^{1/p}$, and with probability $99/100$ we have $E_1 > 1/100$, which implies that $|z_{D(t)}| = O(\|F\|_p)$ for all $t \in [16]$. Conditioned on this, we have $\|z\|_2 < q\|F\|_p$, where q is a constant when $p < 2$, and $q = \Theta(\sqrt{\log(n)})$ when $p = 2$. In either case, we know that the estimate y from count sketch satisfies $\|y - |\zeta|\|_\infty < \epsilon\|\zeta\|_2 < 2\epsilon\|z\|_2 = O(\|F\|_p)$. Thus conditioning on the high probability event that $R = \Theta(\|\zeta\|_2)$, we have that $100\epsilon\mu R = O(\|F\|_p)$, where we can rescale the quantity down by any constant by a suitable rescaling of ϵ.

Now note that $|z_{D(1)}| = \|F\|_p/E_1^{1/p}$ and $|z_{D(1)}| = \|F\|_p/(E_1 + E_2(1 \pm n^{-c+1}))^{1/p}$, where E_1, E_2 are independent exponentials. So with probability $7/8$, we have all of $|z_{D(1)}| = \Theta(\|F\|_p)$, $|z_{D(2)}| = \Theta(\|F\|_p)$, and $|z_{D(1)}| - |z_{D(2)}| = \Theta(\|F\|_p)$ with sufficiently scaled constants, so scaling ν by a sufficiently small constant we have $|\zeta_{D(1)}| = \Theta(\|F\|_p)$, $|\zeta_{D(1)}| - |\zeta_{D(2)}| = \Theta(\|F\|_p)$ and $|\zeta_{D(2)}| = \Theta(\|F\|_p)$. Conditioned on the event in the prior paragraph and on the high probability of success of our L_2 estimation algorithm and our count-sketch error, our estimates of $|\zeta_{D(1)}|, |\zeta_{D(2)}|$ via y are $\Theta(1)$-relative error estimates, so for ϵ small enough the maximum indices in y and ζ will coincide, and we will have both $y_{(1)} - y_{(2)} > 100\epsilon\mu R = O(\|F\|_p)$ and $y_{(2)} > 50\epsilon\mu R = O(\|F\|_p)$. By a union bound, it follows that this condition holds with probability at least $1 - (1/10 + 99/100 + 1/8 + O(n^{-c})) > 1/2$ as desired. ∎

Putting together the results of this section, we obtain the correctness of our algorithm as stated in Theorem 12.

THEOREM 12. *Given any constant $c \geq 2$, $\nu \geq n^{-c}$, and $0 < p \leq 2$, there is a one-pass L_p sampler which returns an index $i \in [n]$ such that $\mathbf{Pr}[i = j] = \frac{|f_j|^p}{\|F\|_p^p}(1 \pm \nu) + n^{-c}$ for all $j \in [n]$, and which fails with probability $\delta > 0$. The space required is $O(\log^2(n)\log(1/\delta)(\log\log n)^2)$ bits for $p < 2$, and $O(\log^3(n)\log(1/\delta)(\log\log n)^2)$ bits for $p = 2$.*

Proof. We claim that conditioned on not failing, we have that $i^* = \arg\max_i\{y_i\} = \arg\max_i\{|z_i|\}$. First, condition on the success of our count-sketch estimator, and on the guarantees of our estimate R, which occur with probability $1 - n^{-c}$ together. Since the gap between the two largest coordinates in y is at least $100\epsilon\mu R > 20\epsilon\|\zeta\|_2 \geq 20\|y - |\zeta|\|_\infty$ (20 times the additive error in estimating $|\zeta|$), it cannot be the case that the index of the maximum coordinate in y is different from the index of the maximum coordinate (in absolute value) in ζ, and moreover both y and ζ must have a unique maximizer. Then we have $|\zeta_{i^*}| - |\zeta_{(2)}| = |\zeta_{(1)}| - |\zeta_{(2)}| > 18\epsilon\|\zeta\|_2$, and since $z_i = (1 \pm O(\nu))\zeta_i$ for all i, we have $\|\,|\zeta| - |z|\,\|_\infty \leq O(\nu)\|\zeta\|_2$. Scaling ν down by a factor of $\epsilon = \Omega(\sqrt{1/\log(n)})$ (which is absorbed into the $O(\nu^{-1})$ update time), the gap between the top two items in ζ is 18 times large than the additive error in estimating z via ζ. Thus we must have $i^* = \arg\max_i\{|\zeta_i\|\} = \arg\max_i\{|z_i|\}$, which completes the proof of the claim.

Now Lemma 12 states for any $i_j \in [n^c]$ that $\mathbf{Pr}[\neg\text{FAIL}\,|\,i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}] = \mathbf{Pr}[\neg\text{FAIL}] \pm O(\log(n)\nu) = q \pm O(\log(n)\nu)$, where $q = \mathbf{Pr}[\neg\text{FAIL}] = \Omega(1)$ is a fixed constant, by Lemma 13, which does not depend on any of the randomness in the algorithm. Since conditioned on not failing we have $\arg\max_i\{y_i\} = \arg\max_i\{|z_i|\}$,

the probability we output $i_j \in [n^c]$ is $\mathbf{Pr}[\neg\text{FAIL} \cap y_{i_j}$ is the max in $y] = \mathbf{Pr}[\neg\text{FAIL} \cap |z_{i_j}|$ is the max in $|z|]$ (conditioned on the high probability events in the prior paragraph), so the probability our final algorithm outputs $i \in [n]$ is

$$\sum_{j \in [n^{c-1}]} \mathbf{Pr}[\neg\text{FAIL} \mid i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}]\mathbf{Pr}[i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}]$$
$$= \sum_{j \in [n^{c-1}]} \frac{|f_i|^p}{\|F\|_p^p}(q \pm O(\log(n)\nu)) = \frac{|f_i|^p}{\|f\|_p^p}(q \pm O(\log(n)\nu)).$$

The potential of the failure of the various high probability events that we conditioned on only adds another additive $O(n^{-c})$ term to the error. Thus, conditioned on an index i being returned, we have $\mathbf{Pr}[i = j] = \frac{|f_j|^p}{\|f\|_p^p}(1 \pm O(\log(n)\nu)) \pm n^{-c}$ for all $j \in [n]$, which is the desired result after rescaling ν down by a factor of $\Omega(1/\log(n))$. (We need only scale down by $\Omega(1/\sqrt{\log(n)})$ after already rescaling by $\epsilon = \Theta(1/\sqrt{\log(n)}$ when $p = 2$.) Running the algorithm $O(\log(\delta^{-1}))$ times in parallel, it follows that at least one index will be returned with probability $1 - \delta$.

Theorem 13 shows that the entire algorithm can be derandomized to use a random seed with $O(\log^2(n)(\log\log n)^2)$-bits for $p < 2$ and $O(\log^3(n)(\log\log n)^2)$-bits for $p = 2$, which dominates the space required to store the sketches of the sampling algorithm themselves. Repeating $O(\log(1/\delta))$ times to obtain δ failure probability gives the stated space bounds. ∎

Remark 2. Using roughly the same update procedures and a similar analysis as in section 5, one can implement the above L_p sampling algorithm to have $\tilde{O}(\nu)$ update time and $\tilde{O}(1)$ report time, just as in Theorem 2. The only difference is the use of Rademacher $\{1, -1\}$ variables in the count-sketch instead of Gaussians and the change to make the variables $g_{i,j,k}$ independent. These Rademacher variables are easier to handle, as one can just compute, for a given bucket $A_{i,j}$ of count-sketch, the number of items which hash into this bucket with a 1 and -1 sign and add the corresponding value to that bucket. This is simply another computation of a binomial random variable. The variables $g_{i,j,k}$ can be handled in `Fast-Update` by modifying the procedure to draw a binomial to determine how many items hash to each bucket $A_{i,j}$ independently for each $j \in [k]$. This is as opposed to the `Fast-Update` of Figure 4, which only allows an item to be hashed into a single bucket in each row of A. In other words, we change Figure 4 to deal with the modified variables $g_{i,j,k}$ by simply removing step 1(d), which decrements the value of W_k, which is the counter of items left to be hashed in a row k of A.

To show that the output of this algorithm is the same when only searching through a subset $\mathcal{K}$ of the coordinates (where $\mathcal{K}$ is as in section 5) for the maximizers $y_{(1)}, y_{(2)}$, observe that the test $y_{(2)} \geq 50\epsilon\mu R$ enforces that, conditioned on not failing, both $y_{(1)}$ and $y_{(2)}$ will be large enough to be contained in the set $\mathcal{K}$. Thus we can safely implement the `Fast-Update` procedure to give improved update time and the `ExpanderSketch` of Theorem 8 to obtain the improved query time.

Appendix B. Derandomizing the original algorithm. We now show how our original algorithm can be derandomized using the same techniques as in section 5. For this section, we let $B \in \mathbb{R}^{O(\log(n))}$ be the sketch stored for the high probability L_2 estimation used in the L_p sampler as in Lemma 11. Note that $B = G \cdot \zeta$, where G is a matrix of i.i.d. Gaussian variables.

THEOREM 13. *The algorithm of section* A.2 *can be derandomized to run in* $O(\log^2(n)\log(1/\delta)(\log\log(n))^2)$ *space for* $p < 2$ *and* $O(\log^3(n)\log(1/\delta)(\log\log(n))^2)$ *space for* $p = 2$.

Proof. We use the same notation $\mathcal{A}(r_e, r_c)$ as in Theorem 7. Recall here that r_e is the randomness required for the exponentials, and r_c is the randomness required for count-sketch (and now r_c must also include the randomness required for the L_2 estimation sketch B). For any fixed randomness r_c, let $\mathcal{A}_{r_c}(r_e)$ be the tester which tests if our L_p sampler would output the index i, where now the bits r_c are hard-coded into the tester, and the random bits r_e for the exponentials are taken as input.

Now note that the entire sketch stored by our algorithm can be written as $Z \cdot \zeta$, where $Z \in \mathbb{R}^{O(\log(n)/\epsilon^2)\times n}$ is a fixed matrix defined by the count-sketch randomness r_c, and ζ is the scaled (by inverse exponentials) and rounded stream vector of the algorithm. Here $Z \cdot \zeta = [\mathtt{vec}(A); \mathtt{vec}(B)]$, where $\mathtt{vec}(A)$ denotes the vectorization of the count-sketch matrix A (and, respectively, B), and $[x; y]$ is vector which stacks x on top of y. Note that we can pull the scalings by F into the matrix Z (making it into a new fixed matrix Z'), so our sketch can be written as $Z' \cdot t$, where for $j \in [n^c]$ we have $t_j = \mathtt{rnd}_\nu(1/t_j^{1/p})$ and t_j's are the i.i.d. exponentials.

Since we are rounding the exponentials to powers of $(1+\nu)$ anyway, we can restrict the support of the coordinates in t to a discrete support of size $O(\mathrm{poly}(n))$ such that each value occurs with probability at least $1/\mathrm{poly}(n)$ for a suitably larger $\mathrm{poly}(n)$. This allows us to sample the variables $\mathtt{rnd}_\nu(1/t_i^{1/p})$ using $O(\log(n))$-bits of space as needed for Lemma 7. Thus our entire algorithm requires $\mathrm{poly}(n)$ random bits to be generated for the exponentials. Similarly, for the random Gaussians used to estimate the L_2 in the sketch B, one can truncate to $O(\log(n))$-bits, incurring only an additive n^{-c} error in these buckets, which can be absorbed into the adversarial error which is already handled in Lemma 12. Restricting the support of the Gaussians so that each value occurs with probability at least $1/\mathrm{poly}(n)$, it follows that these Gaussians can also be sampled using $O(\log(n))$-bits each. The only remaining randomness is the random signs and $h_{i,j,k}$ in count sketch, each of which have a support of size 2 and can be sampled with $O(\log(n))$-bits. So using Lemma 7, we can fool the tester which tests if $Z' \cdot t = y$ for any y with $O(\log(n))$ bounded bit-complexity, using a seed of $O(\log^2(n)(\log\log n)^2)$ bits (and $O(\log^3(n)(\log\log n)^2)$ for $p = 2$). Then as in Theorem 5, since we can fool $\mathbf{Pr}[Z' \cdot t = y]$, we can also fool any tester which takes as input $y = Z' \cdot t$ and outputs whether or not on input y our algorithm would output $i \in [n]$. Thus if $G(x)$ is one instance of the PRG from Lemma 7, we have $\mathbf{Pr}[\mathcal{A}_{r_c}(r_e)] \sim_{n^{-O(\log(n))}} \mathbf{Pr}[\mathcal{A}_{r_c}(G(x))]$, and similarly, as in Theorem 7

$$\begin{aligned}
\mathbf{Pr}\Big[\mathcal{A}(r_e, r_c)\Big] &= \sum_{r_c} \mathbf{Pr}\Big[\mathcal{A}_{r_c}(r_e)\Big]\mathbf{Pr}\Big[r_c\Big] \\
&= \sum_{r_c}((\mathbf{Pr}\Big[\mathcal{A}_{r_c}(G(x))\Big] \pm n^{-O(\log(n))})\mathbf{Pr}\Big[r_c\Big] \\
&= \sum_{r_c}(\mathbf{Pr}\Big[\mathcal{A}_{r_c}(G(x))\Big]\mathbf{Pr}\Big[r_c\Big] \pm \sum_{r_c} n^{-O(\log(n))}\mathbf{Pr}\Big[r_c\Big] \\
&\sim_{n^{-O(\log(n))}} \mathbf{Pr}\Big[\mathcal{A}(G(x), r_c)\Big].
\end{aligned}$$

Now fix any seed $G(x)$, and consider $\mathcal{A}_{G(x)}(r_c)$ which on fixed exponential randomness $G(x)$ and fresh count-sketch randomness r_c, tests whether our algorithm would output $i \in [n]$. Note that this algorithm simply maintains the same sketch

$Z \cdot \zeta = [\mathtt{vec}(A), \mathtt{vec}(B)]$ as above. Note that the entries of Z are of two forms: the i.i.d. count-sketch randomness and the i.i.d. Gaussians needed for the sketch B. By Theorem 5, we can derandomize both of these separately by two more instances $G(x_2), G(x_3)$ of the PRG of Lemma 7, each using seeds x_2, x_3 of $O(\log^2(n)(\log\log n)^2)$ bits of space for $p < 2$ and $O(\log^3(n)(\log\log n)^2)$ bits of space for $p = 2$. So if Z_1 is the first set of rows of Z which correspond to the count-sketch randomness, and Z_2 is the rest of the rows which contain i.i.d. Gaussians, we have that for all y, y' with $O(\log(n))$-entrywise bounded bit complexity, $\mathbf{Pr}[Z_1 \cdot \zeta = y] \sim_{n^{-O(\log(n))}} \mathbf{Pr}[G(x_2) \cdot \zeta = y]$ and $\mathbf{Pr}[Z_2 \cdot \zeta = y'] \sim_{n^{-O(\log(n))}} \mathbf{Pr}[G(x_3) \cdot \zeta = y']$. Here we are abusing notation and thinking of the PRG randomness $G(x_2)$ as being formed into the matrix which it defines.

Since $G(x_2)$ is independent of $G(x_3)$, for any y of $O(\log(n))$-entrywise bounded bit complexity, we have $\mathbf{Pr}[Z \cdot \zeta = y] \sim_{n^{-O(\log(n))}} \mathbf{Pr}[[G(x_2); G(x_3)] \cdot \zeta = y]$. Thus we fool the entire tester $\mathcal{A}_{G(x)}(r_c)$ with $\mathcal{A}_{G(x)}(G(x_2) \cup G(x_3))$, meaning $\mathbf{Pr}[\mathcal{A}_{G(x)}(r_c)] \sim_{n^{-O(\log(n))}} \mathbf{Pr}[\mathcal{A}_{G(x)}(G(x_2) \cup G(x_3))]$, and by a similar averaging arguement as above, we have $\mathbf{Pr}[\mathcal{A}(G(x), r_c)] \sim_{n^{-O(\log(n))}} \mathbf{Pr}[\mathcal{A}(G(x), G(x_2) \cup G(x_3))]$, and thus $\mathbf{Pr}[\mathcal{A}(r_e, r_c)] \sim_{n^{-O(\log(n))}} \mathbf{Pr}[\mathcal{A}(G(x), G(x_2) \cup G(x_3))]$, which completes the proof. We note that any coordinate output by the PRG of Lemma 7 (and thus Theorem 5) can be computed in space linear in the seed length required by Proposition 3, and thus the space required to evaluate the generator is linear in the seed length. ∎

Acknowledgments. The authors would like to thank Raghu Meka for a helpful explanation of the [GKM18] PRG, and for pointing out how the arguments could be extended to fooling functions of multiple half-spaces (Lemma 7). The authors would also like to thank Ryan O'Donnell for a useful discussion on pseudorandom generators in general.

REFERENCES

[AKO10] A. Andoni, R. Krauthgamer, and K. Onak, *Streaming Algorithms from Precision Sampling*, preprint, arXiv:1011.1263, 2010.

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, *Models and issues in data stream systems*, in Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, 2002, pp. 1–16.

[BCIW16] V. Braverman, S. R. Chestnut, N. Ivkin, and D. P. Woodruff, *Beating countsketch for heavy hitters in insertion streams*, in Proceedings of the 48th Annual ACM Symposium on Theory of Computing, ACM, 2016, pp. 740–753.

[BDM02] B. Babcock, M. Datar, and R. Motwani, *Sampling from a moving window over streaming data*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2002, pp. 633–634.

[BKP+14] K. Bringmann, F. Kuhn, K. Panagiotou, U. Peter, and H. Thomas, *Internal DLA: Efficient simulation of a physical growth model*, in Proceedings of the International Colloquium on Automata, Languages, and Programming, Springer, New York, 2014, pp. 247–258.

[BOZ12] V. Braverman, R. Ostrovsky, and C. Zaniolo, *Optimal sampling from sliding windows*, J. Comput. System Sci., 78 (2012), pp. 260–272.

[CCD11] E. Cohen, G. Cormode, and N. G. Duffield, *Structure-aware sampling: Flexible and accurate summarization*, PVLDB, 4 (2011), pp. 819–830.

[CCD12] E. Cohen, G. Cormode, and N. Duffield, *Don't let the negatives bring you down: Sampling from streams of signed updates*, ACM SIGMETRICS Performance Eval. Rev., 40 (2012), pp. 343–354.

[CCFC02a] M. Charikar, K. Chen, and M. Farach-Colton, *Finding frequent items in data streams*, in Proceedings of the 29th International Colloquium on Automata, Languages and Programming, ICALP '02, London, Springer, 2002, pp. 693–703.

[CDK+09] E. Cohen, N. Duffield, H. Kaplan, C. Lund, and M. Thorup, *Stream sampling for variance-optimal estimation of subset sums*, in Proceedings of the 20th Annual

ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2009, pp. 1255–1264.

[CDK+14] E. Cohen, N. G. Duffield, H. Kaplan, C. Lund, and M. Thorup, *Algorithms and estimators for summarization of unaggregated data streams*, J. Comput. System Sci., 80 (2014), pp. 1214–1244.

[CMR05] G. Cormode, S. Muthukrishnan, and I. 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, VLDB Endowment, 2005, pp. 25–36.

[CMYZ10] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, *Optimal sampling from distributed streams*, in Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, 2010, pp. 77–86.

[CMYZ12] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, *Continuous sampling from distributed streams*, J. ACM, 59 (2012), 10.

[Coh15] E. Cohen, *Stream sampling for frequency cap statistics*, in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, 2015, pp. 159–168.

[CT12] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 2012.

[Duf04] N. Duffield, *Sampling for passive internet measurement: A review*, Statist. Sci., 2004, pp. 472–498.

[DV06] A. Deshpande and S. Vempala, *Adaptive sampling and fast low-rank matrix approximation*, in Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, Springer, New York, 2006, pp. 292–303.

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

[FCT15] M. Farach-Colton and M.-T. Tsai, *Exact sublinear binomial sampling*, Algorithmica, 73 (2015), pp. 637–651.

[FIM+01] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright, *Secure multiparty computation of approximations*,in Proceedings of the International Colloquium on Automata, Languages, and Programming, Springer, New York, 2001, pp. 927–938.

[FIS08] G. Frahling, P. Indyk, and C. Sohler, *Sampling in dynamic data streams and applications*, Internat. J. Comput. Geom. Appl., 18 (2008), pp. 3–28.

[FKV04] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM, 51 (2004), pp. 1025–1041.

[GKM18] P. Gopalan, D. M. Kane, and R. Meka, *Pseudorandomness via the discrete Fourier transform*, SIAM J. Comput., 47 (2018), pp. 2451–2487.

[GKMS01] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, *Quicksand: Quick Summary and Analysis of Network Data*, Technical report, 2001.

[GKMS02] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, *How to summarize the universe: Dynamic maintenance of quantiles*, in VLDB'02: Proceedings of the 28th International Conference on Very Large Databases, Elsevier, New York, 2002, pp. 454–465.

[GLH06] R. Gemulla, W. Lehner, and P. J. Haas, *A dip in the reservoir: Maintaining sample synopses of evolving datasets*, in Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, 2006, pp. 595–606.

[GLH08] R. Gemulla, W. Lehner, and P. J. Haas, *Maintaining bounded-size sample synopses of evolving datasets*, VLDB J., 17 (2008), pp. 173–202.

[GM98a] P. B. Gibbons and Y. Matias, *New sampling-based summary statistics for improving approximate query answers*, in Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, WA, 1998, pp. 331–342.

[GMP] P. B. Gibbons, Y. Matias, and V. Poosala, *Fast incremental maintenance of approximate histograms*, NCM Trans Database Syst., 27 (2002).

[Haa81] U. Haagerup, *The best constants in the Khintchine inequality*, Studia Math., 70 (1981), pp. 231–283.

[Haa16] P. J. Haas, *Data-stream sampling: Basic techniques and results*, in Data Stream Management—Processing High-Speed Data Streams, Springer, New York, 2016, pp. 13–44.

[HNG+07] L. Huang, X. L. Nguyen, M. Garofalakis, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, and N. Taft, *Communication-efficient online detection of network-wide*

anomalies, in Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007), IEEE, 2007, pp. 134–142.

[HNSS96] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami, *Selectivity and cost estimation for joins based on random sampling*, J. Comput. System Sci., 52 (1996), pp. 550–569.

[HS92] P. J. Haas and A. N. Swami, *Sequential sampling procedures for query size estimation*, SIGMOD Record, 21 (1992), pp. 341–350.

[Ind06] P. Indyk, *Stable distributions, pseudorandom generators, embeddings, and data stream computation*, J. ACM, 53 (2006), pp. 307–323.

[JST11] H. Jowhari, M. Sağlam, and G. Tardos, *Tight bounds for lp samplers, finding duplicates in streams, and related problems*, in Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '11, ACM, New York, 2011, pp. 49–58.

[JW13] T. S. Jayram and D. P. Woodruff, *Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error*, ACM Trans. Algorithms, 9 (2013), 26.

[KNP+17] M. Kapralov, J. Nelson, J. Pachocki, Z. Wang, D. P. Woodruff, and M. Yahyazadeh, *Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams*, 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), Berkeley, CA, 2017, pp. 475–486.

[Knu98] D. E. Knuth, *The Art of Computer Programming, Volume* II: *Seminumerical Algorithms*, 3rd ed., Addison-Wesley, Reading, MA, 1998.

[KNW10] D. M. Kane, J. Nelson, and D. P. Woodruff, *On the exact space complexity of sketching and streaming small norms*, in Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2010, pp. 1161–1178.

[Kop13] S. Kopparty, *Lecture* 7*: ϵ-biased and Almost k-wise Independent Spaces*, http://sites.math.rutgers.edu/~sk1233/courses/topics-S13/lec7.pdf, 2013.

[LN95] R. J. Lipton and J. F. Naughton, *Query size estimation by adaptive sampling*, J, Comput. System Sci., 51 (1995), pp. 18–25.

[LNNT16] K. G. Larsen, J. Nelson, H. L. Nguyên, and M.Thorup, *Heavy hitters via cluster-preserving clustering*, in Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2016, pp. 61–70.

[LNS90] R. J. Lipton, J. F. Naughton, and D. A. Schneider, *Practical selectivity estimation through adaptive sampling*, in Proceedings of the ACM SIGMOD International Conference on Management of Data, 1990.

[LNW14] Y. Li, H. L. Nguyen, and D. P. Woodruff, *Turnstile streaming algorithms might as well be linear sketches*, in Proceedings of the 46th Annual ACM Symposium on Theory of Computing, ACM, New York, 2014, pp. 174–183.

[M+05] S. Muthukrishnan, *Data streams: Algorithms and applications*, Found. Trends Theor. Comput. Sci., 1 (2005), pp. 117–236.

[McD89] C. McDiarmid, *On the method of bounded differences*, in Surveys in Combinatorics, London Math. Soc. Lecture Note Ser., Cambridge University Press, Cambridge, 1989, pp. 148–188.

[MCS+06] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, *Is sampled data sufficient for anomaly detection?*, in Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, ACM, New York, 2006, pp. 165–176.

[MM12] G. S. Manku and R. Motwani, *Approximate frequency counts over data streams*, PVLDB, 5 (2012), 1699.

[MP14] G. T. Minton and E. Price, *Improved concentration bounds for count-sketch*, in Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2014, pp. 669–686.

[MW10] M. Monemizadeh and D. P. Woodruff, 1*-pass relative-error LP-sampling with applications*, in Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2010, pp. 1143–1160.

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

[Nis92] N. Nisan, *Pseudorandom generators for space-bounded computation*, Combinatorica, 12 (1992), pp. 449–461.

[NZ96] N. Nisan and D. Zuckerman, *Randomness is linear in space*, J. Comput. System Sci., 52 (1996), pp. 43–52.

[Olk93] F. Olken, *Random Sampling from Databases*, Ph.D. thesis, University of California, Berkeley, 1993.

[Pri18] E. Price, *private communication*, 2018.

[TLJ10] M. Thottan, G. Liu, and C. Ji, *Anomaly detection approaches for communication networks*, in Algorithms for Next Generation Networks, Springer, New York, 2010, pp. 239–261.

[TW11] S. Tirthapura and D. P. Woodruff, *Optimal random sampling from distributed streams revisited*, in Proceedings of the International Symposium on Distributed Computing, Springer, New York, 2011, pp. 283–297.

[Vit85a] J. S. Vitter, *Random sampling with a reservoir*, ACM Trans. Math. Software, 11 (1985), pp. 37–57.

[Woo11] D. P. Woodruff, *Near-optimal private approximation protocols via a black box transformation*, in Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, 2011, pp. 735–744.

[WZ13] D. P. Woodruff and Q. Zhang, *Subspace Embeddings and LP-Regression using Exponential Random Variables*, CoRR, abs/1305.5580, 2013.

[WZ16] D. P. Woodruff and P. Zhong, *Distributed low rank approximation of implicit functions of a matrix*, in Proceedings of the 32nd IEEE International Conference on Data Engineering (ICDE), 2016, pp. 847–858.