Annotations for Sparse Data Streams

Amit Chakrabarti[*]   Graham Cormode[†]   Navin Goyal[‡]   Justin Thaler[§]

August 27, 2018

Abstract

Motivated by the surging popularity of commercial cloud computing services, a number of recent works have studied *annotated data streams* and variants thereof. In this setting, a computationally weak *verifier* (cloud user), lacking the resources to store and manipulate his massive input locally, accesses a powerful but untrusted *prover* (cloud service). The verifier must work within the restrictive data streaming paradigm. The prover, who can *annotate* the data stream as it is read, must not just supply the final answer but also convince the verifier of its correctness. Ideally, both the amount of annotation from the prover and the space used by the verifier should be sublinear in the relevant input size parameters.

A rich theory of such algorithms—which we call *schemes*—has started to emerge. Prior work has shown how to leverage the prover's power to efficiently solve problems that have no non-trivial standard data stream algorithms. However, even though optimal schemes are now known for several basic problems, such optimality holds only for streams whose length is commensurate with the size of the *data universe*. In contrast, many real-world data sets are relatively *sparse*, including graphs that contain only $o(n^2)$ edges, and IP traffic streams that contain much fewer than the total number of possible IP addresses, 2^{128} in IPv6.

Here we design the first annotation schemes that allow both the annotation and the space usage to be sublinear in the total number of stream *updates* rather than the size of the data universe. We solve significant problems, including variations of INDEX, SET-DISJOINTNESS, and FREQUENCY-MOMENTS, plus several natural problems on graphs. On the other hand, we give a new lower bound that, for the first time, rules out smooth tradeoffs between annotation and space usage for a specific problem. Our technique brings out new nuances in Merlin–Arthur communication complexity models, and provides a separation between online versions of the MA and AMA models.

1 Introduction

The surging popularity of commercial cloud computing services has rendered the following scenario increasingly plausible. A business—call it AliceSystems—processes billions or trillions of transactions a day. The volume is sufficiently high that AliceSystems cannot or will not store and process the transactions on its own. Instead, it offloads the processing to a commercial cloud computing service.

The offloading of any computation raises issues of trust. AliceSystems may be concerned about relatively benign errors: perhaps the cloud dropped some of the transactions, executed a buggy algorithm, or experienced an uncorrected hardware fault. Alternatively, AliceSystems may be more cautious and fear that the cloud operator is deliberately deceptive or has been externally compromised. Either way, each time

---

[*]Department of Computer Science, Dartmouth College. Supported in part by NSF grant CCF-1217375.

[†]AT&T Labs—Research.

[‡]Microsoft Research India.

[§]School of Engineering and Applied Sciences, Harvard University. Supported by a NSF Graduate Research Fellowship and NSF grants CNS-1011840 and CCF-0915922.

1

AliceSystems poses a query to the cloud, it may demand that the cloud provide not only the answer but also some proof that the returned answer is correct.

Motivated by this scenario, a number of recent works have studied annotated data streams and their variants [7, 9, 10, 11, 21, 24]. In this setting, a computationally weak *verifier* (modeling AliceSystems in the above scenario), who lacks the resources to store the entire input locally, is given access to a powerful but untrusted *prover* (modeling the cloud computing service). The verifier must execute within the confines of the restrictive *data streaming* paradigm, i.e., it must process the input sequentially in whatever order it arrives, using space that is substantially sublinear in the total size of the input. The prover is allowed to annotate the data stream as it is read, with the goal of convincing the verifier of the correct answer. The streaming restriction for the verifier fits the cloud computing setting well, as the verifier's streaming pass over the input can occur while uploading data to the cloud.

Prior work [2, 7, 9, 10, 22, 24] has provided considerable understanding of the power of annotated data streams, revealing a surprisingly rich theory. A number of fundamental problems that possess no non-trivial algorithms in the standard streaming model do have efficient *schemes* when the data stream may be annotated by a prover: the term "scheme" refers to an algorithm involving verifier-prover interaction as above. By exploiting powerful algebraic techniques originally developed in the literature on interactive proofs [18, 26], these works have achieved essentially optimal tradeoffs between annotation size and the space usage of the verifier for problems ranging from frequency moments to bipartite perfect matching.

However, these schemes are only optimal for streams for which the total number of updates is large relative to the size of the *data universe*. In contrast, many real-world data sets are *sparse*: for example, many real-world graphs, though large, contain much fewer than the maximum possible number $\binom{n}{2}$ of edges, and IP traffic streams contain much fewer than the total number of possible IP addresses, 2^{128} in IPv6.

In this paper, we give the first schemes in the annotations model that allow both the annotation size and space usage to be *sublinear in the number of items with non-zero frequency in the data stream*, rather than the size of the data universe n. On the negative side, we also give a new lower bound that for the first time rules out smooth tradeoffs between annotation size and space usage for a specific problem. The latter result is derived from a new lower bound in the Merlin–Arthur (MA) communication model that may be of independent interest.

1.1   Related Work

Aaronson and Wigderson [2] gave a beautiful MA communication protocol for the SET-DISJOINTNESS problem (henceforth, DISJ) using algebraic techniques analogous to those in the famous "sum-check protocol" from the world of interactive proofs and probabilistically checkable proofs [18]. Their protocol is nearly optimal, essentially matching a lower bound of Klauck [22]. The Aaronson–Wigderson protocol has served as the starting point for many schemes for annotated data streams. We will refer to such schemes as *sum-check schemes*; a typical example is Proposition 4.1 in this work.

Aaronson [1] studied the hardness of the INDEX problem in a restricted version of the MA communication model, as well as in a quantum variant of this model. His classical model is similar to the online MA communication model that we consider. Annotated data streams were introduced by Chakrabarti *et al.* [7], and studied further by Cormode *et al.* [9]. These two papers gave essentially optimal annotation schemes for problems ranging from exact computation of Heavy Hitters and Frequency Moments to graph problems like Bipartite Perfect Matching and Shortest *s-t* Path. Cormode, Thaler and Yi [11] later extended the annotations model to allow the prover and verifier to have a *conversation*, and dubbed this interactive model *streaming interactive proofs*. They demonstrated that streaming interactive proofs can have exponentially smaller space and communication costs than annotated data streams, and showed that a number of powerful protocols from the literature on interactive proofs can be made to work with streaming verifiers; in particular, this applies to a powerful general-purpose interactive proof protocol due to Goldwasser, Kalai,

and Rothblum [20]. Cormode, Mitzenmacher, and Thaler [10] implemented a number of protocols in both the annotated data streams and streaming interactive proof settings, demonstrating genuine scalability in many cases. In particular, they developed an implementation of the Goldwasser *et al.* protocol [20] that approaches practicality. Most relevant to our work on annotated data streams, Cormode, Mitzenmacher, and Thaler also used sophisticated FFT algorithms to drastically reduce the prover's runtime in the sum-check schemes, which we make frequent use of.

Two recent works have considered variants of the annotated data stream model. Klauck and Prakash [24] study a restricted version of the annotations model in which the annotation must essentially end by the final stream update. Gur and Raz [21] give protocols for a class of problems in a model that is similar to annotated data streams, but more powerful in that the verifier has access to both public and private randomness. This corresponds to the AMA communication model. We consider protocols in this model in Section 7.2.

Early work on interactive proof systems studied the power of space-bounded verifiers (the survey by Condon [8] provides a comprehensive overview), but many of the protocols developed in this line of work require the verifier to store the input, and therefore do not work in the annotations model, where the verifier must be streaming. An exception is work by Lipton [17], who relied on using fingerprinting techniques to allow a log-space streaming verifier to ensure that the prover correctly plays back the transcript of an algorithm in an appropriate computational model. This approach does not lead to protocols with sublinear annotation length. More recently, Das Sarma *et al.* studied the "best order streaming model," which can be thought of as the annotations model where the annotation is restricted to be a permutation of the input [13].

1.2   Overview of Results and Techniques

We give an informal overview of our results and the techniques we use to obtain them. Throughout, n will denote the size of the data universe and m the number of items with non-zero frequency at the end of a data stream (we refer to m as the "sparsity" of the stream). A scheme in which the streaming verifier uses at most c_v bits of storage and requires at most c_a bits of annotation from the prover is called a (c_a, c_v)-scheme. Section 2 defines our models of computation carefully and sets up terminology.

Section 3 contains our first set of results. We begin by precisely characterizing the complexity of the sparse POINTQUERY problem—a natural variant of the well-known INDEX problem from communication complexity—giving an $(x \log n, y \log n)$-scheme whenever $xy \geq m$. We give similar upper bounds for the related problems SELECTION and HEAVYHITTERS. We also prove a lower bound showing that *any* (c_a, c_v)-scheme for these problems requires $c_a c_v = \Omega(m \log(n/m))$, improving by a $\log(n/m)$ factor over lower bounds that follow from prior work on "dense" streams. By a dense stream we mean one where n is not much larger than m. This $\log(n/m)$ factor may seem minor, but a striking consequence is that the (very) sparse INDEX problem—where Alice's n-bit string has Hamming weight $O(\log n)$—has one-way randomized communication complexity that is within a logarithmic factor of its online MA communication complexity. This implies that no non-trivial tradeoffs between Merlin's and Alice's message sizes are possible for this problem; to our knowledge this is the first problem that provably exhibits this phenomenon.

Our scheme for sparse POINTQUERY relies on universe reduction: the prover succinctly describes a mapping $h : [n] \to [r]$ that maps the input stream, which is defined over the huge data universe $[n]$, down to a derived stream defined over a smaller universe $[r]$. By design, if the prover is honest and the mapping h does not cause "too many collisions," then the answer on the original stream can be determined from the answer on the derived stream. We then efficiently apply known schemes for dense streams to the derived stream.

For our lower bound in Section 3, we give a novel reduction from the standard (dense) INDEX problem to sparse INDEX that is tailored to the MA communication model. We then apply known lower bounds for dense INDEX. Our technique also gives what is to our knowledge the first polynomial separation between the online MA and AMA communication complexities of a specific (and natural) problem.

For clarity, the remainder of this overview omits factors logarithmic in n and m when stating the costs of schemes. Though these factors are important for Section 3 (the consequences of our lower bound being most significant when $n = m^{\omega(1)}$), we anticipate that in practice n and m will usually be polynomially related.

Sections 4 and 5 contain our most interesting and technically involved results, namely, efficient schemes for SIZE-m-SET-DISJOINTNESS (henceforth, m-DISJ) and kth Frequency Moments (henceforth, F_k). The schemes here are substantially more complex than those in Section 3 and represent the main technical contributions of this paper.

Section 4 gives $(m^{2/3}, m^{2/3})$-schemes for both problems, but the schemes rely on "prescient" annotation, i.e., annotation provided at the start of the stream that depends on the stream itself. The even more complex schemes of Section 5 eliminate the need for prescient annotation and also achieve much more general tradeoffs between annotation length and space usage. Specifically, Section 5 gives $(mc_v^{-1/2}, c_v)$-schemes for m-DISJ and F_k for any $c_v < m$. Notice that one recovers the costs achieved in Section 4 by setting $c_v = m^{2/3}$.

These schemes are the first for these problems that allow both the annotation length and space usage to be sublinear in m. At a very high level, there are three interlocking ideas that allow us to achieve this.

1. The first idea is a careful application of universe reduction. We were able to use a simple version of this idea to derive the upper bound for the POINTQUERY problem in Section 3, but in the case of DISJ and F_k the universe-reduction mapping $h : [n] \to [r]$ specified by the prover is more complicated, and requires refinement in the form of the additional ideas described below.

2. The second idea is addressed to ensuring that the prover performed the universe-reduction step in an honest manner, in the sense that the answer on the original stream can indeed be determined from the answer on the derived stream. The difficulty of ensuring P is honest varies depending on the structure of the problem at hand. For F_k, the verifier has to make sure that the universe-reduction mapping h is injective on the items appearing in the data stream. This requires developing an efficient way for V to detect collisions under h, even though V does not have the space to store all of the values $h(x_i)$ for stream updates x_i. For m-DISJ, a notion weaker than injectiveness is sufficient.

3. The third idea pertains to allowing P to specify the universe-reduction mapping h *online*. That is, for many problems it would be much simpler if P could determine the mapping h in advance i.e. if P could be prescient, and send h to V at the start of the stream so that V can determine the derived "mapped-down" stream on her own (this is the approach taken in Section 4). When P must specify h in an online fashion, additional insight is required. At a high level, our approach is to have P specify a "guess" as to the right hash function at the beginning of the steam, and retroactively modify the hash function after the stream has been observed. The challenging aspect of this approach is to ensure that P's retroactive modification of the hash function is consistent with the observed data stream, even though V cannot refer back to the stream to enforce this.

   We exploit similar ideas to allow V to avoid storing the universe-reduction mapping h herself; this is the key to achieving general tradeoffs between annotation length and space usage in Section 5. In some schemes, storing this mapping h would be the bottleneck in V's space usage. We show how V can store only a *partial* description of h, and ask P to fill in the remainder of the description when necessary.

Section 6 exploits all of these results, applying them to several graph problems, including counting triangles and demonstrating a perfect matching. Our schemes have costs that depend on the number of edges in the graph, rather than the total number of possible edges, and demonstrate that the ideas underlying our m-DISJ and F_k schemes are broadly applicable. We state clearly how our schemes improve over prior work throughout.

Section 7 considers a more general stream update model, which allows items to have negative frequencies. These negative frequencies potentially break the "collision detection" sub-protocol used in the previous sections, so we show how to exploit a source of public randomness to allow these protocols to be carried out. Essentially, the public randomness specifies a remapping of the input, so that the prover is highly unlikely to be able to use negative frequencies to "hide" collisions. Because the protocols of Section 7 require public randomness, they work in the AMA communication and streaming models, as opposed to the MA models in which all of our other protocols operate.

2 Models, Notation, and Terminology

Many of the algorithms (schemes) in this paper use randomization in subtle ways, making it important to properly formalize several models of computation. We begin with Merlin–Arthur communication models, a topic first studied by Babai, Frankl and Simon [3], which we eventually use to derive lower bounds. We then turn to annotated data stream models. At the end of the section we set up some notation and terminology for the rest of the paper. Some of our discussion in this section borrows from prior work [7].

2.1 Communication Models

Let $F : X \times Y \to \{0,1\}$ be a function, where X and Y are both finite sets. This naturally gives a 2-player number-in-hand communication problem, where the first player, Alice, holds an input $x \in X$, and the second player, Bob, holds an input $y \in Y$. The players wish to compute $F(x,y)$ by executing a (possibly randomized) communication protocol that correctly outputs $F(x,y)$ with "high" probability. In Merlin–Arthur communication, there is additionally a "super-player," called Merlin, who knows the entire input (x,y), and can help Alice and Bob by interacting with them. The precise pattern of interaction matters greatly and gives rise to distinct models. Merlin's goal is to get Alice and Bob to output "1" regardless of the actual value of $F(x,y)$, and so Merlin is not to be blindly trusted.

One important departure we make from prior work is that *we allow Merlin to use private random coins* during the protocol. Most prior work on MA (and AM) communication [3, 22, 23] defined Merlin to be deterministic, which does not make a difference in the basic setting. But in this work we are concerned with "online MA" models, where the distinction does matter, and these online MA models are in close correspondence with the annotated data stream models that are our eventual topic of study.

MA Communication. In a Merlin–Arthur protocol (henceforth, "MA protocol") for F, Merlin begins by sends a help message $\mathfrak{h}(x,y,r_M)$, using a private random string r_M, that is seen by both Alice and Bob. Then Alice and Bob (the pair that constitutes the entity "Arthur") run a randomized communication protocol $\mathcal{P}$, using a public random string r_A, eventually outputting a bit $\text{out}^{\mathcal{P}}(x,y,r_A,\mathfrak{h})$. Importantly, r_A is not known to Merlin at the time he sends $\mathfrak{h}$. The protocol $\mathcal{P}$ is δ_s-sound and δ_c-complete if there exists a function $\mathfrak{h} : X \times Y \times \{0,1\}^* \to \{0,1\}^*$ such that the following conditions hold.

1. If $F(x,y) = 1$ then $\Pr_{r_M,r_A}[\text{out}^{\mathcal{P}}(x,y,r_A,\mathfrak{h}(x,y,r_M)) = 0] \le \delta_c$.

2. If $F(x,y) = 0$ then $\forall \mathfrak{h}' \in \{0,1\}^* : \Pr_{r_A}[\text{out}^{\mathcal{P}}(x,y,r_A,\mathfrak{h}') = 1] \le \delta_s$.

We define $\text{err}(\mathcal{P})$ to be the minimum value of $\max\{\delta_s, \delta_c\}$ such that the above conditions hold. Following [7], we define the *help cost* $\text{hcost}(\mathcal{P})$ to be $1 + \max_{x,y,r_M} |\mathfrak{h}(x,y,r_M)|$ (forcing hcost ≥ 1, even for traditional Merlin-free protocols), and the *verification cost* $\text{vcost}(\mathcal{P})$ to be the maximum number of bits communicated by Alice and Bob over all x,y and r_A. We define $\text{MA}_\delta(F) = \min\{\text{vcost}(\mathcal{P}) + \text{hcost}(\mathcal{P}) : \mathcal{P}$ is an MA protocol for F with $\text{err}(\mathcal{P}) \le \delta\}$, and $\text{MA}(F) = \text{MA}_{1/3}(F)$.

Online MA Communication. An online MA protocol is defined to be an MA protocol, as above, but with the communication pattern required to obey the following sequence. (1) Input x is revealed to Alice and Merlin; (2) Merlin sends Alice a help message $\mathfrak{h}_1(x, r_M)$ using a private random string r_M; (3) Input y is revealed to Bob; (4) Merlin sends Bob a help message $\mathfrak{h}_2(x, y, r_M)$; (5) Alice sends a public-coin randomized message to Bob, who then gives a 1-bit output. We see this model as the natural MA variant of one-way communication, and the analogy with the gradual revelation of a streamed input should be obvious.

For such a protocol $\mathcal{P}$, we define hcost($\mathcal{P}$) to be $1 + \max_{x,y,r_M}(|\mathfrak{h}_1(x, r_M)| + |\mathfrak{h}_2(x, y, r_M)|)$ We define soundness, completeness, err($\mathcal{P}$), and vcost($\mathcal{P}$) as for MA. Define $\mathrm{MA}_\delta^\to(F) = \min\{\mathrm{hcost}(\mathcal{P}) + \mathrm{vcost}(\mathcal{P}) : \mathcal{P}$ is an online MA protocol for F with err($\mathcal{P}$) $\leq \delta\}$ and write $\mathrm{MA}^\to(F) = \mathrm{MA}_{1/3}^\to(F)$.

Online AMA Communication. An online AMA protocol is a souped-up version of an online MA protocol, where public random coins can be tossed at the start, before any input is revealed. The number of such coin tosses is added to the vcost of the protocol. This models the cost of an initial round of communication between Arthur (i.e., Alice + Bob) and Merlin. Note that the *second* public random string, used when Alice talks to Bob, does not count towards the vcost.

On Merlin's Use of Randomness. In an MA protocol, Merlin can deterministically choose a help message that maximizes Arthur's acceptance probability. However, Merlin cannot do so in the online MA model, because he does not know the entire input when he talks to Alice. This is why we allow Merlin to use randomness in these definitions.

Two recent papers [7, 24] use "online MA" to mean a more restrictive model where a deterministic Merlin talks only to Bob and not to Alice. With Merlin required to be deterministic, this communication restriction is irrelevant, as Merlin cannot tell Alice anything she does not already know. However, we permit Merlin to be probabilistic, and in this case we do not know that Merlin can avoid talking to Alice.

As noted earlier, our goal in defining the communication models this way is to closely correspond to annotated data stream models. In many of our online schemes (see, e.g., Section 5), the helper provides initial annotation that specifies a random "hash" function, h, and the completeness guarantee of the subsequent protocol depends crucially on h having "low collision" properties. Since h must be chosen without seeing all of the input, such low collision properties cannot be guaranteed by picking a fixed h in advance. However, if the helper chooses h at random, then we do have such guarantees for each fixed input, with high probability.

2.2 Data Stream Models

We now define our annotated data stream models. Recall that a (traditional) data stream algorithm computes a function F of an input sequence $\mathbf{x} \in \mathcal{U}^N$, where N is the number of stream updates, and $\mathcal{U}$ is some data universe, such as $\{0,1\}^b$ or $[n] = \{0, \ldots, n-1\}$: the algorithm uses a limited amount of working memory and has access to a random string. The function F may or may not be Boolean.

An annotated data stream algorithm, or a *scheme*, is a pair $\mathcal{A} = (\mathfrak{h}, V)$, consisting of a help function $\mathfrak{h} : \mathcal{U}^N \times \{0,1\}^* \to \{0,1\}^*$ used by a *prover* (henceforth, P) and a data stream algorithm run by a *verifier*, V. Prover P provides $\mathfrak{h}(\mathbf{x}, r_P)$ as annotation to be read by V. We think of $\mathfrak{h}$ as being decomposed into $(\mathfrak{h}_1, \ldots, \mathfrak{h}_N)$, where the function $\mathfrak{h}_i : \mathcal{U}^N \to \{0,1\}^*$ specifies the annotation supplied to V after the arrival of the ith token x_i. That is, $\mathfrak{h}$ acts on $\mathbf{x}$ (using r_P) to create an *annotated stream* $\mathbf{x}^{\mathfrak{h}, r_P}$ defined as follows:

$$\mathbf{x}^{\mathfrak{h}, r_P} := (x_1, \mathfrak{h}_1(\mathbf{x}, r_P), x_2, \mathfrak{h}_2(\mathbf{x}, r_P), \ldots, x_N, \mathfrak{h}_N(\mathbf{x}, r_P)).$$

Note that this is a stream over $\mathcal{U} \cup \{0,1\}$, of length $N + \sum_i |\mathfrak{h}_i(\mathbf{x}, r_P)|$. The streaming verifier V, who uses w bits of working memory and has oracle access to a (private) random string r_V, then processes this annotated stream, eventually giving an output $\mathrm{out}^V(\mathbf{x}^{\mathfrak{h}, r_P}, r_V)$.

Prescient Schemes. The scheme $\mathcal{A} = (\mathfrak{h}, V)$ is said to be δ_s-sound and δ_c-complete for the function F if the following conditions hold:

1. For all $\mathbf{x} \in \mathcal{U}^N$, we have $\Pr_{r_P, r_V}[\text{out}^V(\mathbf{x}^{\mathfrak{h}, r_P}, r_V) \neq F(\mathbf{x})] \leq \delta_c$.

2. For all $\mathbf{x} \in \mathcal{U}^N$, $\mathfrak{h}' = (\mathfrak{h}'_1, \mathfrak{h}'_2, \ldots, \mathfrak{h}'_N) \in (\{0,1\}^*)^N$, we have $\Pr_{r_V}[\text{out}^V(\mathbf{x}^{\mathfrak{h}'}, r_V) \notin \{F(\mathbf{x})\} \cup \{\bot\}] \leq \delta_s$.

If $\delta_c = 0$, the scheme satisfies *perfect completeness*; otherwise it has *imperfect completeness*. An output of "$\bot$" indicates that V rejects P's claims in trying to convince V to output a particular value for $F(\mathbf{x})$.

We note two important things. First, the definition of a scheme allows the annotation $\mathfrak{h}_i(\mathbf{x}, r_P)$ to depend on the entire stream $\mathbf{x}$, thus modeling *prescience*: the advice from the prover can depend on data which the verifier has not seen yet. Second, P must convince V of the value of $F(\mathbf{x})$ *for all* $\mathbf{x}$. This is stricter than the traditional definitions of interactive proofs and MA communication complexity (including our own, above) for decision problems, which place different requirements on the cases $F(\mathbf{x}) = 0$ and $F(\mathbf{x}) = 1$. In Section 6, we briefly consider a relaxed definition of schemes that is in the spirit of the traditional definition.

We define $\text{err}(\mathcal{A})$ to be the minimum value of $\max\{\delta_s, \delta_c\}$ such that the above conditions are satisfied. We define the *annotation length* $\text{hcost}(\mathcal{A}) = \max_{\mathbf{x}, r_P} \sum_i |\mathfrak{h}_i(\mathbf{x}, r_P)|$, the total size of P's communications, and the *verification space cost* $\text{vcost}(\mathcal{A}) = w$, the space used by the verifier V. We say that $\mathcal{A}$ is a prescient (c_a, c_v)-scheme if $\text{hcost}(\mathcal{A}) = O(c_a)$, $\text{vcost}(\mathcal{A}) = O(c_v)$ and $\text{err}(\mathcal{A}) \leq \frac{1}{3}$.

Online Schemes. We call $\mathcal{A} = (\mathfrak{h}, V)$ a δ-error online scheme for F if, in addition to the conditions in the previous definition, each function $\mathfrak{h}_i$ depends only on $(x_1, \ldots, x_i)$. We define error, hcost, and vcost as above and say that $\mathcal{A}$ is an *online (c_a, c_v)-scheme* if $\text{hcost}(\mathcal{A}) = O(c_a)$, $\text{vcost}(\mathcal{A}) = O(c_v)$, and $\text{err}(\mathcal{A}) \leq \frac{1}{3}$.

Unlike prior work [7], we do not always assume that the universe size n and stream length N are polynomially related; it is possible that $\log N = o(\log n)$. Therefore we must be much more careful about logarithmic factors than in prior work. We do assume that $N < n$ always, because our focus is on sparse streams.

Notice that the help function can be made deterministic in a prescient scheme, but not necessarily so in an online scheme. This is directly analogous to the situation for MA and online MA communication models, as discussed at the end of Section 2.1.

AMA Schemes. We also consider what we call AMA schemes, where there is a common source of public randomness, in addition to the verifier's private random coins. The AMA scheme model is identical to the one considered by Gur and Raz [21], who referred to it as the "Arthur–Merlin streaming model."

An online AMA scheme is identical to a (standard) online scheme, except that the data stream algorithm and help function both have access to a source of public random bits. The number of random bits used is also counted in both the hcost and the vcost of the scheme.

On Practicality and the Plausibility of Prescience. Although our definition of a scheme allows annotation to be sent after each stream update, all the schemes we in fact design in this paper only require annotation before the start or after the end of the stream. As a practical matter, this avoids the need for fine-grained coordination between the annotation and the data stream.

Online annotation schemes have the appealing property that the prover need not "see into the future" to execute them; at any time t, the prover's message only depends on stream updates that arrived before time t. While the online restriction appears most natural, prescient schemes may still be suitable in some settings, such as when P has already seen the full input prior to V beginning to read it. Consider a volunteer computing scenario where the verifier farms out many computations to volunteers, and only inspects a particular input if a volunteer has already looked at that input and claims to have found something interesting[1]. In brief, in some settings the prover may naturally see the input before the verifier, and in this case a prescient scheme will be feasible.

---

[1]See, for example, `http://boinc.berkeley.edu/`.

2.3 Relationship Between MA Protocols and Schemes

Any prescient (resp. online) (c_a, c_v)-scheme $\mathcal{A} = (\mathfrak{h}, V)$ for a function F can be converted into an MA (resp. online MA) protocol for F in the natural way: Merlin sends the output of the ith help function $\mathfrak{h}_i$ to Alice—who receives a prefix of the input stream—or Bob, depending on which of the players possesses the ith piece of the input. Alice runs the streaming algorithm V on her input as well as any annotation she received, and sends the state of the algorithm to Bob. Bob uses this state to continue running V on his input and the annotation he received, and then outputs the end result. The hcost of this protocol is at most $c_a \log N$, since Merlin has to specify which stream update i each piece of annotation is associated with, and the vcost of this protocol is at most c_v. Thus, lower bounds on usual (resp. online) MA communication protocols imply related lower bounds on the costs of prescient (resp. online) annotated data stream algorithms.

2.4 Additional Notation and Terminology

A data stream specifies an input $\mathbf{x}$ incrementally. Typically, $\mathbf{x}$ can be thought of as a vector (although more generally it may represent a graph or a matrix). Each update in the stream is of the form (i, δ) where $i \in \mathcal{U}$ identifies an element of the universe, and $\delta \in \mathbb{Z}$ describes the change to the frequency of i. The frequency of universe item i is defined as $f_i(\mathbf{x}) := \sum_{(j_k, \delta_k) \in \mathbf{x}: j_k = i} \delta_k$. We refer to the vector $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ as the *frequency vector* of $\mathbf{x}$, where n denotes the size of the data universe.

We consider several different update models. In the most general update model, the *non-strict turnstile model*, the δ values may be negative, and so f_i may also be negative. In the *strict turnstile model*, the δ values may be negative, but it is assumed that the frequencies f_i always remain non-negative. In the *insert-only model*, the δ values must be non-negative. Orthogonal to these, in the *unit-update* version of each model, the δ values are assumed to have absolute value 1. Each of our results applies to a subset of these models, and we specify within the statement of each theorem which update models it applies to.

Throughout, n will denote the size of the data universe, N will denote the total number of stream updates, m will denote the total number of items with non-zero frequency at the end of the stream, and M will refer to the total number of distinct items that ever appear within some stream update. We will refer to N as the *length* of the stream, to m as *sparsity* of the stream, and to M as the *footprint* of the stream. Notice that it is always the case that $m \leq M \leq N$. In the case of insert-only streams, $m = M$, but for streams in the (strict or general) turnstile models it is possible for m to be much smaller than M. Note also that while we talk about "sparse" streams, this refers to the relative size of n and m, not the absolute size. Indeed, we assume that m is typically large, too large for V to store the stream explicitly (else the problems can become trivial).

We often make use of *fingerprint* functions of streams, which enable a streaming verifier to test whether two large streams have the same frequency vector. The verifier chooses a fingerprint function $g(\mathbf{x})$ at random from some family of functions satisfying the property that (over the random selection of the function g),

$$\Pr[g(\mathbf{x}) = g(\mathbf{y}) \mid f(\mathbf{x}) \neq f(\mathbf{y})] < 1/p$$

for a parameter p. Typically, $g(\mathbf{x})$ is an element of a finite field of size $\text{poly}(p)$, and hence the number of bits required to store the value $g(\mathbf{x})$ (as well as g itself) is $O(\log p)$. Further, there are known constructions of fingerprint functions where $g(\mathbf{x})$ can be computed in space $O(\log p)$ by a streaming algorithm in the non-strict turnstile update model [7].

3 Point Queries, Index, Selection, and Heavy Hitters

3.1 Upper Bounds

Our first result is an efficient online annotation scheme for the POINTQUERY problem, a generalization of the familiar INDEX problem.

Definition 3.1. In the POINTQUERY problem, the data stream $\mathbf{x}$ consists of a sequence of updates of the form (i, δ), followed by an index ι, and the goal is to determine the frequency $f_\iota(\mathbf{x}) = \sum_{(j_k, \delta_k) \in \mathbf{x}: j_k = \iota} \delta_k$.

A prescient $(\log n, \log n)$-scheme for this problem is trivial as P can just tell V the index ι at the start of the stream, and V can track the frequency of ι while observing the stream. The vcost can be improved to $O(\log m)$ if V retains a hashed value of ι, and tracks the frequency of matching updates. The first scheme has perfect completeness, while the second has completeness error polynomially small in m.

The costs of the scheme below are in terms of the stream sparsity m, and not the stream length N or the stream footprint M; this is significant if $m \ll M$, which is the case, e.g., for the well-known straggler and set-reconciliation problems that have been studied in traditional streaming and communication models [14, 19]. Our lower bound in Theorem 3.9 shows our scheme is essentially optimal for moderate universe sizes, i.e. when the universe size n is sub-exponential in the sparsity m.

Theorem 3.2. *For any pair (c_a, c_v) such that $c_a \cdot c_v \geq m$, there is an online $(c_a \log n, c_v \log n)$-scheme in the non-strict turnstile update model for the* POINTQUERY *problem with imperfect completeness. Any online (c_a, c_v) scheme with $c_a \geq \log n$ for this problem requires $c_a \cdot c_v = \Omega(m \log(n/m))$.*

Proof. V requires P to specify at the start of the stream a hash function $h : [n] \to [c_v]$. V requires h to have description length $O(c_a)$, rejecting if this is not the case. We define the derived streams $\mathbf{x}^j \in \mathcal{U}^N$ based on h: we set $\mathbf{x}^j_k = x_k$ iff $h(x_k) = j$, and 0 otherwise. Intuitively, the hash function h partitions the stream updates in $\mathbf{x}$ into c_v disjoint buckets, and the vector $\mathbf{x}^j$ describes the contents of the jth bucket. V maintains fingerprints over a field of size $\text{poly}(n)$ of each of the c_v different $\mathbf{x}^j$ vectors.

At the end of the stream, given the desired index ι, P provides a description of the (claimed) frequency vector in the $h(\iota)$th derived stream, $f(\mathbf{x}^{h(\iota)})$. V computes a fingerprint of the claimed frequency vector, and compares it to the fingerprint she computed from the data stream, accepting if and only if the fingerprints match. Since each $\mathbf{x}^j$ is sparse in expectation, the cost of this description can be low: provided h does not map more than $O(c_a)$ items with non-zero frequency to $h(\iota)$, P can just specify the item id and frequency of the items with non-zero frequency in $f(\mathbf{x}^{h(\iota)})$. In this case, the annotation size is just $O(c_a \log n)$. If P exceeds this amount of annotation, V will halt and reject (output $\bot$).

Soundness follows from the fingerprinting guarantee: if P does not honestly provide $\mathbf{x}^{h(\iota)}$, V's fingerprint of $\mathbf{x}^{h(\iota)}$ computed from the data stream will not match her fingerprint of the claimed vector of frequencies.

To show (imperfect) completeness, we study the probability that the output of an honest prover is rejected. This happens only if $m(\mathbf{x}^{h(\iota)})$, the number of non-zero entries in $\mathbf{x}^{h(\iota)}$, is much larger than its expectation. By the pairwise independence of h, $\mathbb{E}[m(\mathbf{x}^{h(\iota)})] = m(\mathbf{x})/c_v = c_a$. Thus, by Markov's inequality, $\Pr[m(\mathbf{x}^{h(\iota)}) > 10 c_a] < 1/10$. So by specifying a hash function chosen at random from a pairwise independent hash family, and then honestly playing back the items that map to the same region as ι, P can convince V to accept with probability $9/10$.

Notice that V does not need to *enforce* that P picks the hash function h at random from a pairwise-wise independent hash family, as P has no incentive not to pick the hash functions in this way. That is, since V will reject if too many items map to the same region as ι, it is *sufficient* for P to pick h at random from a pairwise independent hash family in order to convince V to accept with constant probability. But it is equally acceptable if P wants to pick h another way; if he does so, P just risks that V will reject with a higher probability.

The lower bound follows from Theorem 3.9, which we prove in Section 3.2. $\qquad\square$

The scheme of Theorem 3.2 yields nearly optimal schemes for the HEAVYHITTERS and SELECTION problems, described below. Table 1 summarizes these results and compares to prior work.

9

Problem	Scheme Costs	Completeness	Prescience	Source
POINTQUERY	$(\log n, \log n)$	Perfect	Prescient	[7]
POINTQUERY	$(m\log n, \log n)$	Perfect	Online	[7]
POINTQUERY	$(c_a\log n, c_v\log n)\colon c_ac_v \geq n$	Perfect	Online	[7]
POINTQUERY	$(c_a\log n, c_v\log n)\colon c_ac_v \geq m$	Imperfect	Online	Theorem 3.2
SELECTION	$(c_a\log n, c_v\log n)\colon c_ac_v \geq n$	Perfect	Online	[7]
SELECTION	$(m\log n, \log n)$	Perfect	Online	[7]
SELECTION	$(c_a\log^2 n, c_v\log n)\colon c_ac_v \geq m\log n$	Imperfect	Online	Corollary 3.4
ϕ-HEAVYHITTERS	$(\phi^{-1}\log n, \phi^{-1}\log n)\colon c_ac_v \geq n$	Perfect	Prescient	[7]
ϕ-HEAVYHITTERS	$(\phi^{-1}c_a\log n, c_v\log n)\colon c_ac_v \geq n$	Perfect	Online	[7]
ϕ-HEAVYHITTERS	$(m\log n, \log n)$	Perfect	Online	[7]
ϕ-HEAVYHITTERS	$(\phi^{-1}c_a\log n, c_v\log n)\colon c_ac_v \geq m\log n$	Imperfect	Online	Corollary 3.6
ϕ-HEAVYHITTERS	$(\phi^{-1}\log n + c_a\log n, c_v\log n)\colon c_ac_v \geq m\log n$	Imperfect	Online	Corollary 5.6

Table 1: Comparison of our schemes to prior work. For all three problems, ours are the first online schemes to achieve both annotation and space usage sublinear in the stream sparsity m when $m \ll \sqrt{n}$, and we strictly improve over the online MA communication cost of prior schemes whenever $m = o(n)$. For brevity, we omit factors of $\log_{c_v}(m)$ from the statement of costs of the ϕ-HEAVYHITTERS scheme due to Corollary 5.6

3.1.1 Selection

Our definition of the SELECTION problem assumes all frequencies $f_i := \sum_{(j_k,\delta_k):j_k=i} \delta_k$ are non-negative, and so this definition is only valid for the strict turnstile update model.

Definition 3.3. The SELECTION problem is defined in terms of the quantity $N = \sum_{i\in[n]} f_i$, the sum of all the frequencies. Given a desired rank $\rho \in [N]$, output an item j from the stream $\mathbf{x} = \langle(j_1,\delta_1),\ldots,(j_m,\delta_m)\rangle$, such that $\sum_{(j_k,\delta_k):j_k<j} \delta_k < \rho$ and $\sum_{(j_k,\delta_k):j_k>j} \delta_k \geq N-\rho$.

Corollary 3.4. *For any pair (c_a,c_v) such that $c_ac_v \geq m\log n$, there is an online $(c_a\log^2 n, c_v\log n)$-scheme for* SELECTION *in the strict turnstile update model.*

The corollary follows from a standard observation to reduce SELECTION to answering prefix sum queries, and hence to multiple instances of the POINTQUERY problem. V treats each stream update (i,δ) in the stream $\mathbf{x}$ as an update to $O(\log n)$ dyadic ranges, where a dyadic range is a range of the form $[j2^k,(j+1)2^k-1]$ for some j and k. Thus, we can view the set of dyadic range updates implied by $\mathbf{x}$ as a derived stream of sparsity $m\log n$. Notice we are using the fact that this transformation from the original stream of sparsity m results in a derived stream of sparsity at most $m\log n$; a different derived stream was used in [7] to address the SELECTION problem, but the sparsity of that derived stream could be substantially larger than the sparsity of the original stream.

For any i, the quantity $T_i := \sum_{(j,\delta):j\leq i} \delta$ can be written as the sum of the counts of $O(\log n)$ dyadic ranges. Thus, at the end of the stream P can convince V that item i has the desired T_i value by running $\log n$ POINTQUERY protocols as in Theorem 3.2 in parallel on the derived stream of sparsity $m\log n$. The verifier's space usage is the same as for a single POINTQUERY instance on this stream: V fingerprints each of the derived streams $\mathbf{x}^j$ defined in the proof of Theorem 3.2, and uses these fingerprints in all $\log n$ instances of the POINTQUERY scheme. The annotation length is $\log n$ times larger than that required for a single POINTQUERY instance because P may have to describe the frequency vectors of up to $\log n$ derived streams.

Thus, we get an online $(c_a\log^2 n, c_v\log n)$-scheme as long as $c_ac_v = \Omega(m\log n)$.

3.1.2 Frequent Items

Our definition of the ϕ-HEAVYHITTERS problem also assumes all frequencies $f_i := \sum_{(j_k,\delta_k):j_k=i}\delta_k$ are non-negative, and so this definition is only valid for the strict turnstile update model.

Definition 3.5. The ϕ-HEAVYHITTERS problem (also known as frequent items) is to list those items i such that $f_i \geq \phi N$, i.e. whose frequency of occurrence exceeds a ϕ fraction of the total count $N = \sum_{i\in[n]} f_i$.

We give a preliminary result for the ϕ-HEAVYHITTERS problem in Corollary 3.6 below. We give a substantially improved scheme in Section 5 using the ideas underlying our online scheme for frequency moments.

Corollary 3.6. *For all c_a, c_v such that $c_a c_v \geq m\log n$, there is an online $(c_a\phi^{-1}\log n, c_v\log n)$-scheme for solving ϕ-HEAVYHITTERS in the strict turnstile update model.*

Corollary 3.6 follows from the following analysis. [7, Theorem 6.1] describes how to reduce ϕ-HEAVYHITTERS to demonstrating the frequencies of $O(\phi^{-1})$ items in a derived stream. Moreover, the derived stream has sparsity $O(m\log n)$ if the original stream has sparsity m. We use the POINTQUERY scheme of Theorem 3.2. As in Corollary 3.4, the annotation length blows up by a factor ϕ^{-1} relative to a single POINTQUERY, but the space usage of V can remain the same as in a single POINTQUERY instance. Hence, we obtain an online $(c_a\phi^{-1}\log n, c_v\log n)$-scheme for any $c_a c_v \geq m\log n$.

3.2 Lower Bound

In this section, we prove a new lower bound on the online MA communication complexity of the (m,n)-Sparse INDEX problem.

Definition 3.7. In the (m,n)-Sparse INDEX problem, Alice is given a vector $x \in \{0,1\}^n$ of Hamming weight at most m, and Bob is given an index ι. Their goal is to output the value x_ι.

We prove our lower bound by reducing the (dense) INDEX problem (i.e. the (m,n)-Sparse INDEX problem with $m = \Theta(n)$) in the MA communication model to the (m,n)-Sparse INDEX problem for small m. The idea is to replace Alice's dense input with a sparser input over a bigger universe, and then take advantage of our sparse POINTQUERY protocol. A lower bound on the online MA communication complexity of the dense INDEX problem was proven in [7, Theorem 3.1]; there, it was shown that any online MA communication protocol $\mathcal{P}$ requires $\mathrm{hcost}(\mathcal{P})\mathrm{vcost}(\mathcal{P}) \geq n$. Combining this with our reduction of the dense INDEX problem to the sparse version, we conclude that any protocol for sparse INDEX must be costly.

Lemma 3.8. *[7, Theorem 3.1] Any online MA communication protocol $\mathcal{P}$ for the (n,n)-Sparse INDEX problem must have $\mathrm{hcost}(\mathcal{P})\mathrm{vcost}(\mathcal{P}) = \Omega(n)$.*

Remark 1. The lower bound of Lemma 3.8 was originally proved by Chakrabarti *et al.* [7] in the communication model in which Merlin cannot send any message to Alice. However, the proof easily extends to our online MA communication model (where Merlin can send a message to Alice, but that message cannot depend on Bob's input).

Theorem 3.9. *Any online MA communication protocol $\mathcal{P}$ for the (m,n)-Sparse INDEX problem for which $\mathrm{hcost}(\mathcal{P}) \geq \log n$ must have $\mathrm{hcost}(\mathcal{P})\mathrm{vcost}(\mathcal{P}) = \Omega(m\log(n/m))$.*

Proof. Assume we have an online MA communication protocol $\mathcal{P}$ for (m,n)-sparse INDEX. We describe how to use this online MA protocol for the sparse INDEX problem to design one for the dense INDEX problem on vectors of length $n' = m\log(n/m)$.

Let $k = \log(n/m)$. Given an input x to the dense INDEX problem, Alice partitions x into n'/k blocks of length k, and constructs a 0-1 vector y of Hamming weight n'/k over the universe $\{0,1\}^{(n'/k)\cdot 2^k} = \{0,1\}^n$ as follows. She replaces each block B_i with a 1-sparse vector $v_i \in \{0,1\}^{2^k}$, where each entry of v_i corresponds to one of the 2^k possible values of block B_i. That is, if block B_i of x equals the binary representation of the number $j \in [2^k]$, then Alice replaces block B_i with the vector $e_j \in \{0,1\}^{2^k}$, where e_j denotes the vector with a 1 in coordinate j and 0s elsewhere.

Alice now has an $n'/k = m$-sparse derived input y over the universe $\{0,1\}^n$. Merlin looks at Bob's input to see what is the index ι of the dense vector x that Bob is interested in. Merlin then tells Bob the index ℓ such that $\ell = 2^k(\iota - 1) + j$, where B_i is the block that ι is located in, and block B_i of Alice's input x equals the binary representation of the number $j \in [2^k]$. Notice that Merlin can specify ℓ using $\log n$ bits. If Bob is convinced that $y_\ell = 1$, then Bob can deduce the value of *all* the bits in block B_i of the original dense vector x, and in particular, the value of x_ι.

The parties then run the assumed online MA protocol for (m,n)-Sparse INDEX. The total hcost of this protocol is $\mathrm{hcost}(\mathcal{P}) + \log n = O(\mathrm{hcost}(\mathcal{P}))$, and the total vcost is $\mathrm{vcost}(\mathcal{P})$. Thus, by Lemma 3.8, $\mathrm{hcost}(\mathcal{P})\,\mathrm{vcost}(\mathcal{P}) = \Omega(n') = \Omega(m\log(n/m))$ as claimed. $\qquad\square$

Theorem 3.9 should be contrasted with the following well-known upper bound.

Theorem 3.10. *Assume $n < m^m$. Then the one-way randomized communication complexity of the (m,n)-Sparse INDEX Problem is $O(m\log m)$.*

Proof. Alice chooses a hash function $h : [n] \to [m^3]$ at random from a pairwise independent family and uses h to perform "universe reduction". That is, she sends h along with the set S of m values $\{h(j) : x_j = 1\}$. Notice h can be specified with $O(\log n) = O(m\log m)$ bits, and S can be specified with $O(m\log m)$ bits. Bob outputs 1 if $h(\iota) \in S$, and 0 otherwise. The correctness of the protocol follows from the pairwise independence property of h: if $x_\iota = 0$, then with high probability ι will not collide under h with any j such that $x_j = 1$. The total cost of this protocol is $O(m\log m)$. $\qquad\square$

3.3 Implications of the Lower Bound

Our lower bound in Theorem 3.9 has interesting consequences when it is combined with the upper bound in Theorem 3.10. Consider in particular the (m,n)-Sparse INDEX Problem, where $n = 2^m$. Theorem 3.10 implies that the one-way randomized communication complexity of this problem is $O(m\log m)$; that is, without any need of Merlin, Alice and Bob can solve the problem with $O(m\log m)$ communication.

Meanwhile, Theorem 3.9 implies that even if Merlin's message to Bob has length $\Omega(\log n) = \Omega(m)$, Alice's message to Bob must have length $\Omega(m\log(n/m)/m) = \Omega(m)$. Indeed, Theorem 3.9 shows that for any protocol $\mathcal{P}$, if $\mathrm{hcost}(\mathcal{P}) \geq \log n = m$, then we must have $\mathrm{hcost}(\mathcal{P})\,\mathrm{vcost}(\mathcal{P}) = \Omega(m\log(n/m)) = \Omega(m^2)$. In particular, this means that if $\mathrm{hcost}(\mathcal{P}) = m$, $\mathrm{vcost}(\mathcal{P})$ must be $\Omega(m)$. This trivially implies that for any protocol $\mathcal{P}$ with $\mathrm{hcost}(\mathcal{P})$ *less* than m, $\mathrm{vcost}(\mathcal{P})$ must still be $\Omega(m)$; otherwise we could achieve a protocol with $\mathrm{hcost}(\mathcal{P}) = m$ and $\mathrm{vcost}(\mathcal{P}) = o(m)$ simply by running $\mathcal{P}$ and adding in extraneous bits to the proof to bring the proof length up to m.

Consequently, the online MA communication complexity of this problem is at least $\Omega(m)$, which is at most a logarithmic factor smaller than the one-way randomized communication complexity. To our knowledge, this is the first problem that provably exhibits this behavior. Specifically, this rules out smooth tradeoffs between annotation size and space usage in any annotated streaming protocol for the $(m, 2^m)$-Sparse INDEX Problem.

Corollary 3.11. *The one-way randomized communication complexity of the $(m, 2^m)$-Sparse INDEX Problem is $O(m\log m)$. The online Merlin-Arthur communication complexity is $\Omega(m)$.*

3.3.1 Other Sparse Problems

A number of lower bounds in [7] are proved via reductions from INDEX that preserve stream length up to logarithmic factors. This holds for SELECTION and HEAVYHITTERS, as well as for the problem of determining the existence of a triangle in a graph. For all such problems, the lower bound of Theorem 3.9 implies corresponding new lower bounds for sparse streams, i.e. streams for which $m = o(n)$. We omit the details for brevity.

3.3.2 Separating Online MA and AMA Communication Complexity

Another implication of Theorem 3.9 is a polynomial separation between online MA communication complexity and online AMA communication complexity. Indeed, there is an online AMA protocol of cost $\tilde{O}(\sqrt{m})$ for the $(m, 2^{\sqrt{m}})$-Sparse INDEX Problem, where the $\tilde{O}$ notation hides factors polylogarithmic in m: the first message, which consists of public random coins, is used to specify a hash function $h : [n] \to [m^3]$ from a pairwise independent hash family; this message has length $O(\log n) = O(\sqrt{m})$. With high probability, h is injective on the set $\{j : x_j = 1\}$. The parties then run the online MA communication protocol of Theorem 3.2 on the inputs $h(\mathbf{x})$ and $h(\iota)$ and output the result. The total cost of this protocol is $\tilde{O}(\sqrt{m})$ as claimed. In Appendix A, we in fact show that up to logarithmic factors in m, this online AMA protocol is optimal.

Meanwhile, the lower bound of Theorem 3.9 implies that the online MA communication complexity of this problem is $\Omega(m^{3/4})$. Indeed, if we have a protocol $\mathcal{P}$ with hcost$(\mathcal{P}) = m^{3/4} > \log n$, Theorem 3.9 implies that hcost$(\mathcal{P})$ vcost$(\mathcal{P}) = \Omega(m \log(n/m)) = \Omega(m^{3/2})$, and hence vcost$(\mathcal{P}) > m^{3/4}$.

To our knowledge, this is the first such separation between online AMA and online MA communication complexity (we remark that polynomial separations between online MA and MAMA communication complexity were already known, for problems including INDEX and DISJ [2, 7]). Indeed, all previous lower bound methods that apply to online MA communication complexity, such as the proof of [7, Theorem 3.1] and the methods of Klauck and Prakash [24], in fact yield equivalent AMA lower bounds. At a high level, the reason is that these methods work via round reduction – they remove the need for Merlin's message. They therefore turn any online MA protocol for a function F into an online "A" protocol for F, which is really just a one-way randomized protocol without a prover, allowing one to invoke a known lower bound on the one-way randomized communication complexity of F. Similarly, they turn an online AMA protocol for F into an online AA protocol, which is also just a one-way randomized protocol for F.

The reason Theorem 3.9 is capable of separating online AMA from MA communication complexity is that the reduction in the proof of Theorem 3.9 turns an online MA protocol for the (m, n)-Sparse INDEX Problem into an online MA protocol for the (dense) INDEX Problem with related costs. However, the natural variant of the reduction applied to an online AMA protocol for the (m, n)-Sparse INDEX Problem yields an online MAMA protocol for the dense INDEX Problem, *not* an online AMA protocol (see Appendix A for details). And the dense INDEX Problem has an online MAMA protocol that is polynomially more efficient than any online AMA protocol (see e.g. [2, 11]).

4 Prescient Schemes for Sparse Disjointness and Frequency Moments

In this section and the next, we describe schemes for the m-Disjointness (m-DISJ) and Frequency Moment (F_k) problems. These schemes contain the main ideas of the paper.

Scheme Costs	Completeness	Prescience	Source
$(m\log m)^{2/3}, (m\log m)^{2/3})$: $m = \Omega(\log n)$	Perfect	Prescient	Theorem 4.3
$(c_a \log n, c_v \log n)$: $c_a c_v \geq n$	Perfect	Online	[7]
$(m\log n, \log n)$	Perfect	Online	[7]
$(c_a \log n \log_{c_v} m, c_v \log n \log_v m)$: $c_a = mc_v^{-1/2}$	Imperfect	Online	Theorem 5.1

Table 2: Comparison of our m-DISJ schemes to prior work. Ours are the first schemes to achieve annotation length and space usage that are both sublinear in m for $m \ll \sqrt{n}$, and we strictly improve over the MA communication cost (online or prescient) of prior schemes whenever $m = o(n)$.

4.1 Background: Optimal Schemes for Dense Problems

We begin with a scheme achieving optimal tradeoffs between annotation length and space usage for a broad class of dense problems. Though this scheme follows readily from prior work [7, 9], we describe it in detail for completeness. This scheme is a good example of a *sum-check scheme* as described in Section 1.1, and is based on the Aaronson–Wigderson MA protocol for DISJ [2].

Proposition 4.1. *Let $f^{(1)}, \ldots, f^{(\ell)}$ denote the frequency vectors of ℓ data streams, each over the universe $[n]$. Let g be an ℓ-variate polynomial of total degree d over the integers. Let $F = \sum_{i=1}^{n} g(f_i^{(1)}, \ldots, f_i^{(\ell)})$, and let o be an a priori upper bound on $|F|$. Then for positive integers c_a, c_v with $c_a c_v \geq n$, there is an online $(dc_a(\log n + \log o), \ell c_v(\log n + \log o))$-scheme for computing F in the non-strict turnstile update model.*

Proof. We work on $\mathbb{F}_q$, the finite field with q elements, for a suitably large prime q; the choice $q > 2d(n+o)^2$ suffices. V treats each n-dimensional vector $f^{(j)}$ as a $c_a \times c_v$ array with entries in $\mathbb{F}_q$, using any canonical bijection between $[c_a] \times [c_v]$ and $[n]$, and interpreting integers as elements of $\mathbb{F}_q$ in the natural way. Through interpolation, this defines a unique bivariate polynomial $\tilde{f}^{(j)}(X,Y) \in \mathbb{F}_q[X,Y]$ of degree $c_a - 1$ in X and $c_v - 1$ in Y, such that for all $x \in [c_a]$, $y \in [c_v]$, $\tilde{f}^{(j)}(x,y) = f^{(j)}(x,y)$.

The polynomials $\tilde{f}^{(j)}$ can then be evaluated at locations outside $[c_a] \times [c_v]$, so in the scheme V picks a random position $r \in \mathbb{F}_q$, and evaluates $\tilde{f}^{(j)}(r,y)$ for all $j \in [\ell]$ and $y \in [c_v]$; V can do this using c_v words of memory per vector $f^{(j)}$ in a streaming manner [7, Theorem 4.1]. Let $\tilde{g}$ denote the total-degree-d polynomial over $\mathbb{F}_q$ that agrees with g at all inputs in $\mathbb{F}_q^{\ell}$. P then presents a polynomial $b(X)$ of degree at most $d(c_a - 1)$ that is claimed to be identical to $\sum_{y \in [c_v]} \tilde{g}(\tilde{f}^{(1)}(X,y), \ldots, \tilde{f}^{(\ell)}(X,y))$.

V checks that $b(r) = \sum_{y \in [c_v]} \tilde{g}(\tilde{f}^{(1)}(r,y), \ldots, \tilde{f}^{(\ell)}(r,y))$. If this *sum check* passes, then V believes P's claim and accepts $\sum_{x \in [c_a]} b(x)$ as the correct answer. It is evident that this scheme satisfies perfect completeness. The proof of soundness follows from the Schwartz-Zippel lemma: if P's claim is false, then

$$\Pr\left[b(r) = \sum_{y \in [c_v]} \tilde{g}\left(\tilde{f}^{(1)}(r,y), \ldots, \tilde{f}^{(\ell)}(r,y) \right) \right] \leq d(c_a - 1)/q. \qquad \square$$

4.2 A Prescient Scheme for Sparse Disjointness

An important special case of the communication problem DISJ is when Alice's and Bob's input sets are promised to be small, i.e., have size at most $m \ll n$. These should be thought of as *sparse* instances. The sparsity parameter m has typically been denoted by the letter k in the communication complexity literature, and the problem has typically been referred to as k-DISJ rather than m-DISJ; we use m rather than k for consistency with our notation in the rest of the paper (where m denotes the sparsity of a data stream).

Among the original motivations for studying this variant is its relation to the clique-vs.-independent-set problem introduced by Yannakakis [27] to study linear programming formulations for combinatorial optimization problems. More recent motivations include connections to property testing [4]. A clever protocol

of Håstad and Wigderson [16] gives an optimal $O(m)$ communication protocol for m-DISJ, improving upon the trivial $O(m \log n)$ and the easy $O(m \log m)$ bounds. This protocol requires considerable interaction between Alice and Bob, a feature that turns out to be necessary. Recent results of Buhrman *et al.* [6] and Dasgupta *et al.* [12] give tight $\Theta(m \log m)$ bounds for m-DISJ in the one-way model. Very recently, Brody *et al.* [5] and Sağlam and Tardos [25] have given tight rounds-vs.-communication tradeoffs for m-DISJ.

Here we obtain the first nontrivial bounds for m-DISJ in the annotated streams model, and thus also in the online MA communication model.

Definition 4.2. In the m-DISJ problem, the data stream specifies two multi-sets $S, T \subseteq [n]$, with $\|S\|_0, \|T\|_0 \leq m$, where $\|S\|_0$ denotes the number of distinct items in S. An update of the form $((0, i), \delta)$ is interpreted as an insertion of δ copies of item i into set S, and an update of the form $((1, i), \delta)$ is interpreted as an insertion of δ copies of item i into T. The goal is to determine whether or not S and T are disjoint.

Notice Definition 4.2 allows S and T to be multi-sets, but assumes the strict turnstile update model, where the frequency of each item is non-negative.

Theorem 4.3. *Assume $m > \log n$. There is a prescient $((m \log m)^{2/3}, (m \log m)^{2/3})$-scheme for m-DISJ with perfect completeness in the strict turnstile update model. In particular, the MA-communication complexity of m-DISJ is $O((m \log m)^{2/3})$. Any prescient (c_a, c_v) protocol requires $c_a c_v = \Omega(m)$.*

Proof. Obviously if S and T are not disjoint, the prescient prover can provide an item $i \in S \cap T$ at the start of the stream and the verifier can check that i indeed appears in both S and T. The total space usage and annotation length is just $O(\log n)$ in this case.

Suppose now that S and T are disjoint. We first recall that a $(\sqrt{n} \log n, \sqrt{n} \log n)$-scheme for DISJ follows from Proposition 4.1, with $f^{(1)}$ and $f^{(2)}$ set to the indicator vectors of S and T respectively, and g equal to the product function. We refer to this as the dense DISJ scheme because its cost does not improve if $|S|$ and $|T|$ are both $o(n)$.

Our prescient scheme for m-DISJ works as follows. At the start of the stream, the prover describes a hash function $h : [n] \to [r]$, for some smaller universe $[r]$, with the property that h is injective on $S \cup T$. We will write $h(S)$ to denote the result of applying h to every member of S. The parties can now run the dense DISJ scheme whereby P convinces V that $h(S)$ and $h(T)$ are disjoint. Given the existence of an injective function h, perfect completeness follows from the fact that if S and T are disjoint, so are $h(S)$ and $h(T)$, combined with the perfect completeness of the dense DISJ scheme. Soundness follows from the fact that if $i \in S \cap T$, then $h(i) \in h(S) \cap h(T)$ i.e. if S and T are not disjoint, then the same holds trivially for $h(S)$ and $h(T)$.

The dense DISJ scheme run on $h(S)$ and $h(T)$ requires annotation length and space usage $O(\sqrt{r} \log r)$. We now show that, for a suitable choice of r, P's description of h is also limited to $O(\sqrt{r} \log r)$ communication, balancing out the cost of the rest of the scheme.

A family of functions $\mathcal{F} \subseteq [r]^{[n]}$ is said to be κ-perfect if, for all $S \subseteq [n]$ with $|S| \leq \kappa$, there exists a function $h \in \mathcal{F}$ that is injective when restricted to S. Fredman and Komlós [15] have shown that for all $n \geq r \geq \kappa$, there exists a κ-perfect family $\mathcal{F}$, with

$$|\mathcal{F}| \leq (1 + o(1)) \left(\frac{\kappa \log n}{-\log(1 - t(r, \kappa))} \right),$$

where

$$t(r, \kappa) := \prod_{j=1}^{\kappa-1} \left(1 - \frac{j}{r} \right).$$

For $r \geq 2\kappa$, we can use the crude approximation

$$-\log(1 - t(r, \kappa)) \geq t(r, \kappa) \geq \left(1 - \frac{\kappa}{r} \right)^\kappa \geq e^{-2\kappa^2/r}$$

15

Scheme Costs	Completeness	Prescience	Source
$(k^2 c_a \log n,\ kc_v \log n)$: $c_a c_v \geq n$	Perfect	Online	[7]
$(m \log n,\ \log n)$	Perfect	Online	[7]
$(k^2 m^{2/3} \log n,\ km^{2/3} \log n)$	Perfect	Prescient	Theorem 4.5
$(k^2 m c_v^{-1/2} \log n \log_{c_v} m,\ kc_v \log n \log_{c_v} m)$: $c_v > 1$	Imperfect	Online	Theorem 5.1

Table 3: Comparison of our F_k schemes to prior work. Ours are the first schemes to achieve annotation length and space usage that are both sublinear in m for $m \ll \sqrt{n}$, and we strictly improve over the MA communication cost of prior protocols (online or prescient) whenever $m = o(n)$.

to obtain the bound $|\mathcal{F}| = O(\kappa e^{2\kappa^2/r} \log n)$, which implies

$$\log |\mathcal{F}| \ = \ O(\kappa^2/r),$$

for $\kappa^2/r = \Omega(\log \kappa)$ and $\kappa = \Omega(\log n)$.

Let us pick a family $\mathcal{F}$ that is $(2m)$-perfect. Once P and V agree upon such a family $\mathcal{F}$, the prover, upon seeing the input sets S and T, can pick $h \in \mathcal{F}$ that is injective on $S \cup T$. Describing h requires $O(m^2/r)$ bits; P sends this to V before the stream is seen, and V stores it while observing the stream in order to run the dense DISJ scheme on $h(S)$ and $h(T)$. To balance out this communication with the $O(\sqrt{r} \log r)$ cost of running the dense DISJ scheme on $h(S)$ and $h(T)$, we choose r so that

$$\frac{m^2}{r} \ = \ \Theta(\sqrt{r} \log r).$$

This is achieved by setting $r = m^{4/3}/\log^{2/3} m$. The resulting upper bound is that both the annotation length and verifier's space usage are $O\left((m \log m)^{2/3}\right)$.

The lower bound follows from known lower bounds for dense streams [7]. $\qquad\square$

4.3 A Prescient Scheme for Frequency Moments

We now present prescient schemes for the kth Frequency Moment problem, F_k.

Definition 4.4. In the F_k problem, the data stream **x** consists of a sequence of updates of the form (i, δ), and the frequency of item i is defined to be $f_i = \sum_{(j_\ell, \delta_\ell) \in \mathbf{x}: j_\ell = i} \delta_\ell$. The goal is to compute $F_k = \sum_{i \in [n]} f_i^k$.

The idea behind the scheme, as in the case of m-DISJ, is that P is supposed to specify a "hash function" h to reduce the universe size in a way that does not introduce false collisions. However, for F_k it is essential that V ensure h is truly injective on the items appearing in the data stream. This is in contrast to m-DISJ, where a weaker notion than injectiveness was sufficient to guarantee soundness. The fundamental difference between the two problems is that for m-DISJ, collisions only "hurt the prover's claim" that the two sets are disjoint, whereas for F_k the prover could try to use collisions to convince the verifier that the answer to the query is higher or lower than the true answer.

Theorem 4.5. *There is a prescient $(k^2 m^{2/3} \log n, km^{2/3} \log n)$-scheme for computing F_k over a data stream of sparsity m in the strict turnstile update model. This scheme has perfect completeness. Any prescient (c_a, c_v) protocol requires $c_a c_v = \Omega(m)$.*

Proof. The idea is to have the prover specify for the verifier a perfect hash function $h : [n] \to [r]$, where r is to be determined later, i.e. P specifies a hash function h such that for all $x \neq y$ appearing in at least one update in the data stream, $h(x) \neq h(y)$. The verifier stores the description of h, and while observing the stream runs

the dense F_k scheme of Proposition 4.1 on the derived stream in which each update (i,δ) is replaced with the update $(h(i),\delta)$.

As discussed above, it is essential that V ensure h is injective on the set of items that have non-zero frequency, as otherwise P could try to introduce collisions to try to trick the verifier. To deal with this, we introduce a mechanism by which V can "detect" collisions.

Definition 4.6. Define the problem INJECTION as follows. We observe a stream of tuples $t_i = ((x_i,b_i),\delta_i)$. Each t_i indicates that δ_i copies of item x_i are placed in bucket $b_i \in [r]$. We allow δ_i to be negative, modeling deletions, and refer to the quantity $f_{(j,b)} = \sum_{i:(x_i,b_i)=(j,b)} \delta_i$ as the *count* of pair (j,b). We assume the strict turnstile model, so that for all pairs (j,b) we have $f_{(j,b)} \geq 0$.

We say that the stream is an injection if for every two pairs (j,b) and (j',b) with positive counts, it holds that $j = j'$. Define the output as 1 if the stream defines an injection, and 0 otherwise.

Lemma 4.7. *For any $c_a c_v \geq r$, there is an online $(c_a \log r, c_v \log r)$-scheme for determining whether a stream in the strict turnstile model is an injection.*

Proof. Say that bucket b is *pure* if there is at most one $j \in [n]$ such that $f_{(j,b)} > 0$. The stream defines an injection if and only if every bucket b is pure.

Notice that a bucket b is pure if and only if the variance of the item identifiers mapping to the bucket with positive count is zero. Intuitively, our scheme will compute the sum of the these variances across all buckets b; this sum will be zero if and only if the stream defines an injection. Details follow.

Define three r-dimensional vectors u, v, w as follows:

$$u_b = \sum_{j \in [n]} f_{(j,b)},$$

$$v_b = \sum_{j \in [n]} f_{(j,b)} j,$$

$$w_b = \sum_{j \in [n]} f_{(j,b)} j^2.$$

It is easy to see that if bucket b is pure then $v_b^2 = u_b \cdot w_b$. Moreover, if bucket b is impure then $v_b^2 < u_b w_b$; this holds by the Cauchy-Schwarz inequality applied to the n-dimensional vectors whose jth entries are $\sqrt{f_{(j,b)}}$ and $\sqrt{f_{(j,b)}} \cdot j$ respectively (the strict inequality holds because for an impure bucket b, the vector given by $\sqrt{f_{(j,b)}} \cdot j$ is not a scalar multiple of the vector given by $\sqrt{f_{(j,b)}}$). Here, we are exploiting the assumption that $f_{(j,b)} \geq 0$ for all pairs (i,b), as this allows us to conclude that all $\sqrt{f_{(j,b)}}$ values are real numbers.

It follows that $\sum_{b \in [r]} v_b^2 = \sum_{b \in [r]} u_b \cdot w_b$ if and only if the stream defined an injection. Both quantities can be computed using the "dense" scheme of Proposition 4.1. Notice that each update $t_i = ((x_i,b_i),\delta_i)$ contributes independently to each of the vectors u, v, and w, and hence it is possible for V to run the scheme of Proposition 4.1 on these vectors as required. This yields an online $(c_a \log r, c_v \log r)$-scheme for the injection problem for any $c_a c_v \geq r$ as claimed. $\square$

Returning to our F_k scheme, P specifies a hash function h claimed to be one-to-one on the set of items that appear in one or more updates of the stream $\mathbf{x}$. V verifies that h is injective using the scheme of Lemma 4.7. If this claim is true, then $F_k(\mathbf{x}) = F_k(h(\mathbf{x}))$, the frequency moment of the mapped-down stream, and P can prove this by running the scheme of [7, Theorem 4.1] on the derived stream $h(\mathbf{x})$.

Perfect completeness follows from P's ability to find a perfect hash function just as in Theorem 4.3. Soundness follows from the soundness of the INJECTION scheme of Lemma 4.7, in addition to the soundness property of the F_k scheme of [7, Theorem 4.1].

To analyze the costs, note that by using the hash family of Fredman and Komlós [15], the annotation length and space cost due to specifying and storing the hash function h is $O(m^2 \log n/r)$. The annotation length and space cost of the dense F_k scheme of Proposition 4.1 are $O(k^2 c_a \log r)$ and $O(k c_v \log r)$ for any $c_a c_v \geq r$. The annotation length and space cost of the INJECTION scheme can be set to $O(c_a \log r)$ and $O(c_v \log r)$ respectively. Setting $r = m^{4/3}$ and $c_a = c_v = m^{2/3}$ yields the desired costs. $\quad\square$

5 An Online Scheme for Frequency Moments

We now give an online version of F_k scheme of Theorem 4.5. A simple modification of this scheme yields the scheme for m-DISJ with analogous costs as claimed in Row 4 of Table 2. In addition to avoiding the use of prescience, our online scheme avoids requiring V to explicitly store the hash function sent by P, allowing us to achieve a much wider range of tradeoffs between annotation size and space usage relative to Theorems 4.3 and 4.5.

Theorem 5.1. *For any $c_v > 1$, there is an online $(k^2 m c_v^{-1/2} \log n \log_{c_v} m,\ k c_v \log n \log_{c_v} m)$-scheme for F_k in the strict turnstile model for a stream of sparsity m over a universe of size n. Any online (c_a, c_v)-scheme for this problem with $c_a \geq \log n$ requires $c_a c_v = \Omega(m \log(n/m))$.*

Notice that the annotation length is less than $m \log n$ for any $c_v = m^{\Omega(1)}$, and therefore this protocol is not subsumed by the simple "sparse" scheme (second row of Table 3) in which P just replays the entire stream in a sorted order, and V checks this is done correctly using fingerprints. Notice also that the product of the space usage and annotation length is $k^3 m c_v^{1/2} \log^2 n \log_{c_v}^2 m$, which is in $o(n)$ for many interesting parameter settings. This improves upon the dense sum-check scheme (first row of Table 3) in such cases.

5.1 An Overview of the Scheme

In order to achieve an online scheme, we examine how to construct perfect hash functions such as those used in the prescient F_k scheme of Theorem 4.5. Let S be the set of m items with non-zero frequency at the end of the stream: we want the hash function to be one-to-one on S. Choose a hash function h at random from pairwise independent hash family mapping $[n]$ to $[r]$, for r to be specified later – this requires just $O(\log n)$ bits to specify. We only expect $O(m^2/r)$ pairs to collide under h, which means that with constant probability there will be $O(m^2/r)$ collisions if h is chosen as specified. The final hash function h^* is specified by writing down h (which takes only $O(\log n)$ bits), followed by the items involved in a collision and some special locations for them. The total (expected) bit length to specify this hash function is $O(m^2 \log(n)/r)$.

In our online F_k scheme, P will send such an h at the start of the stream. Notice h does not depend on the stream itself – it is just a random pairwise independent hash function – so P is not using prescience. P also has no incentive not to choose h at random from a pairwise independent hash family, since the only purpose of choosing h in this manner is to minimize the number of collisions under h. If P chooses h in a different way, P simply risks that there are too many collisions under h, causing V to reject.

Now while V observes the stream, she runs the online sum-check scheme for F_k given in Proposition 4.1 on the mapped-down universe of size r, using h as the mapping-down function. At the end of the stream, P is asked to retroactively specify a hash function h^* that is one-to-one on S as follows. P provides a list L_0 of all items in S that were involved in a collision under h, accompanied by their frequencies. Assuming that these items and their frequencies are honestly specified by P, V can compute their contribution to F_k and *remove them* from the stream. By design, h^* is then (claimed to be) injective on the remaining items. V can confirm this tentatively using the INJECTION scheme of Lemma 4.7.

The remainder of the scheme is devoted to making the correctness a certainty by ensuring that the items in L_0 and their frequencies are as claimed (we stress that while our exposition of the scheme is modular, all

parts of the scheme are executed in parallel, with no communication ever occurring from V to P). A naive approach to checking the frequencies of the items in L_0 would be to run $|L_0|$ independent POINTQUERY schemes, one for each item in L; however there are too many items in L_0 for this to be cost-effective. Instead, we check all of the frequencies as a batch, with a (sub-)scheme whose cost is roughly equal to that of a single INJECTION query.

This (sub-)scheme can be understood as proceeding in stages, with each stage i using a different pairwise independent hash function h_i to map down the full original input. Say that an item j is *isolated* by h_i if j is not involved in a collision under h_i with any other item with non-zero frequency in the original data stream $\mathbf{x}$. The goal of stage i is to isolate a large fraction of items which were not isolated by any previous stage.

A key technical insight is that at each stage i, it is possible for V to "ignore" all items that are not isolated at that stage. This enables V to check that the frequencies of all items that *are* isolated at stage i are as claimed. We bound the number of stages that are required to isolate all items if P behaves as prescribed – if P reaches an excessive number of stages, then V will simply reject.

5.2 Details of the Scheme

Proof of Theorem 5.1: Let $r = mc_v^{1/2}$. P sends a hash function $h : [n] \to [r]$ at the start of the stream, claimed to be chosen at random from a pairwise independent hash family. While observing the stream, V runs the dense online sum-check scheme for F_k given in Proposition 4.1 on the mapped-down universe $[r]$. Let S be the set of items with non-zero frequency at the end of the stream. After the stream is observed, P is asked to provide a list L_0 of all items with nonzero frequency that were involved in a collision, followed by a claimed frequency f_i^* for each $i \in L_0$.

Assuming that these items and their frequencies are honestly specified in L_0 by P, V can compute their contribution $C_0 = \sum_{i \in L_0} f_i^*$ to F_k and then remove them from the stream by processing updates $U = \{(i, -f_i^*) : i \in L_0\}$ within the dense F_k scheme. h is injective on the remaining items. V can confirm this using the INJECTION scheme of Lemma 4.7 (conditioned on the assumed correctness of L_0). Thus the dense F_k scheme will output $C_1 = \sum_{i \notin L_0} f_i^k$. Assuming all of V's checks within the dense F_k scheme pass, V outputs $C_0 + C_1$ as the answer.

The remainder of the scheme is directed towards determining that the frequency of items in L_0 are correctly reported. We abstract this goal as the following problem.

Definition 5.2. Define the ℓ-MULTIINDEX problem as follows. Consider a data stream $\mathbf{x} \circ L_0$, where $\circ$ denotes concatenation. $\mathbf{x}$ is a usual data stream in the strict turnstile model, while L_0 is a list of ℓ pairs (i, f_i^*). Let f be the frequency vector of $\mathbf{x}$. The desired output is 1 if $f_i = f_i^*$ for all $i \in L_0$, and 0 otherwise.

We defer our solution to the ℓ-MULTIINDEX problem to Section 5.3. For now, we state our main result about the problem in the following lemma.

Lemma 5.3. *For all $c_v > 1$, ℓ-MULTIINDEX has an online $(mc_v^{-1/2} \log n \log_{c_v} \ell, \; c_v \log n \log_{c_v} \ell)$-scheme in the strict turnstile update model.*

Analysis of Costs. Let S be the set of items with non-zero frequency when the stream ends. First, we argue that if r is the size of the mapped-down universe, and P chooses the hash function h at random from a pairwise independent hash family, then with probability $9/10$, there will be at most $10m^2/r$ items in S that collide under g. Indeed, by a union bound, the probability any item i with non-zero count is involved in a collision is at most m/r, and hence by linearity of expectation, the expected number of items involved in a collision is at most m^2/r.

So by Markov's inequality, with probability at least $9/10$, the total number of items involved in a collision will be at most $10m^2/r = O(mc_v^{-1/2})$ under the setting $r = mc_v^{1/2}$. Conditioned on this event, P can specify

the list L_0 and the associated frequencies with annotation length $O(mc_v^{-1/2}\log n)$, and V can use the MULTI-INDEX scheme of Lemma 5.3 with $\ell = O(mc_v^{-1/2})$ to verify the frequencies of the items in L_0 are as claimed. For any $c_v > 1$, Lemma 5.3 under this setting of ℓ yields an $(mc_v^{-1}\log n \cdot \log_{c_v} \ell, c_v \log n \cdot \log_{c_v} \ell)$-scheme.

Running all of the sum-check schemes (i.e., the INJECTION scheme and the F_k scheme itself) on the mapped-down universe requires annotation $O(k^2 rc_v^{-1}\log r)$ and space $O(kc_v \log r)$ for V; in total, this provides an online $(m^2 \log n/r + k^2 r \log n/c_v + kmc_v^{-1}\log n \cdot \log_{c_v} m, c_v \log n \cdot \log_{c_v} M)$-scheme.

Since we set $r = mc_v^{1/2}$, we obtain a online $(k^2 mc_v^{-1/2}\log_{c_v}(m), kc_v \log n \log_v(m))$-scheme for any $c_v > 1$.

The lower bound stated in Theorem 5.1 follows from Theorem 3.9 and an easy reduction from the (m,n)-sparse INDEX problem. $\qquad\square$

5.3 A Scheme for MultiIndex: Proof of Lemma 5.3

Before presenting an efficient online scheme for the ℓ-MULTIINDEX Problem, we define two "sub"problems, which apply a function to only a subset of the desired input.

Definition 5.4. Define the problem SUBINJECTION as follows. We observe a stream of tuples $t_i = (x_i, b_i, \delta_i)$, followed by a vector $z \in \{0,1\}^r$. As in the INJECTION problem, each t_i indicates that δ_i copies of item x_i are placed in bucket $b_i \in [r]$.

We say that the stream defines a *subinjection* based on z if for every b such that $z_b \geq 1$, for every two pairs (x,b) and (y,b) with positive counts, it holds that $x = y$. The SUBINJECTION problem is to decide whether the stream defines a subinjection based on z.

Notice that the INJECTION problem is a special case of the SUBINJECTION problem with $z_i = 1$ for all i.

Lemma 5.5. *For any $c_a c_v \geq r$, there is an online $(c_a \log r, c_v \log r)$-scheme for SUBINJECTION in the strict turnstile update model. Moreover, for any constant $c > 0$, this scheme can be instantiated to have soundness error $1/r^c$.*

Proof. Define vectors u, c_v, and w exactly as in the proof of Lemma 4.7, and observe that the stream defines a sub-injection if and only if $\sum_{b\in[r]} z_b v_b^2 = \sum_{b\in[r]} z_b u_b w_b$. V can compute both quantities using the dense scheme of Proposition 4.1, with the same asymptotic costs as the scheme of Lemma 4.7. The soundness error can be made smaller than $1/r^c$ for any constant c by running the scheme of Proposition 4.1 over a finite field of size $\mathrm{poly}(r)$, for a sufficiently fast-growing polynomial in r. $\qquad\square$

We similarly define the problem SUBF_2 over a data universe of size n based on a vector $z \in \{0,1\}^n$ as $\sum_{i\in[n]} z_i f_i^2$, the sum of squared frequencies of items indicated by z. This too is a low-degree polynomial function of the input values, and so Proposition 4.1 implies SUBF_2 can be computed by an online $(c_a \log r, c_v \log r)$-scheme in the general turnstile update model for any c_a, c_v such that $c_a c_v \geq r$ (and the soundness error in this protocol can be made smaller than $1/r^c$ for any desired constant c).

Online scheme for ℓ-MULTIINDEX. The scheme can be thought of as proceeding in t stages (t will be specified later), although these stages merely serve to partition the annotation: there is no communication from V to P during these stages. Each stage j makes use of a corresponding hash function $h_j : [n] \to [r]$ for $r = mc_v^{1/2}$. The t hash functions are provided by P at the start of the stream, so that V has access to them throughout the stream. Each h_j is claimed to be chosen at random from a pairwise independent hash family: if they are, then there are unlikely to be too many collisions, so P has no incentive not to choose h_j at random. Let f denote the vector of frequencies defined by the input stream, and let $f^{(0)}$ denote the vector satisfying $f_i^{(0)} = f_i$ for $i \in L_0$, and $f_i^{(0)} = 0$ for $i \notin L_0$.

Stage j begins with a list L_{j-1} of items. We will refer to these items as "exceptions". P provides a new list $L_j \subseteq L_{j-1}$ of items which remain exceptions in stage j; P implicitly claims that no items in $L_{j-1} \setminus L_j$

collide with some other input items under hash function h_j. Let $z^{(j)}$ denote the indicator vector of the list of buckets corresponding to $L_{j-1} \setminus L_j$, i.e. $z^{(j)}_{h_j(i)} = 1$ if $i \in L_{j-1} \setminus L_j$, and $z^{(j)}$ entries are 0 otherwise. To check that no items in $L_{j-1} \setminus L_j$ collide under h_j, V will use the SUBINJECTION scheme based on the indicator vector $z^{(j)}$ over the full original input f as mapped by the hash function h_j. Note that since the original input stream is in the strict turnstile update model, so is the stream on which the SUBINJECTION scheme is run (as the SUBINJECTION scheme is simply run on the original input stream as mapped by the hash function h_j, based on the vector $z^{(j)}$). Note also that L_{j-1} and L_j are provided explicitly, so V can compute $z^{(j)}$ easily.[2]

Having established that the items in $L_{j-1} \setminus L_j$ are no longer exceptions, V also wants to ensure that the frequencies of these items were reported correctly in L_0. To do so, V run the SUBF$_2$ scheme over the vector $f - f^*$ as mapped by h_j to r buckets, based on the $z^{(j)}$ indicator vector. The result is zero if and only if $f_i = f_i^{(j)}$ for all i where $z_i^{(j)} = 1$.

The stages continue until $L_j = \emptyset$, and there are no more exceptions. Provided all schemes conclude correctly, and the number of stages to reach $L_j = \emptyset$ is at most t, V can accept the result, and output 1 for the answer to the MULTIINDEX decision problem.

Lastly, note that V does not need to explicitly store any of the lists L_j. In fact, P can implicitly specify all of the lists L_j while playing the list L_0: for each item $i \in L_0$, he provides a number j, thereby implicitly claiming that $i \in L_{j'}$ for $j' \le j$, and $i \notin L_{j'}$ for $j' > j$.

Analysis of costs. If h_j is chosen at random from a pairwise independent hash family, the probability an item i in L_{j-1} is involved in a collision with the original stream f under h_j is $O(m/r) = O(c_v^{-1/2})$. Consider the probability that any item i survives as an exception to stage t. The probability of this is $O(c_v^{-t/2})$, and summed over all ℓ items, the expected number is $O(\ell c_v^{-t/2})$. Invoking Markov's inequality, with constant probability it suffices to set $t = O(\log_{c_v} \ell)$ to ensure that we need at most t stages before no more exceptions need to be reported.

In stage j, the SUBINJECTION and SUBF$_2$ schemes cost $(mc_v^{-1/2} \log n, c_v \log n)$. Summing over the t stages, we achieve for any $c_v > 1$ an $(mc_v^{-1/2} \log(n) \cdot \log_{c_v}(m), c_v \log(n) \cdot \log_{c_v}(m))$-scheme as claimed in the statement of Lemma 5.3.

Formal Proof of Soundness. The soundness error of the protocol can be bounded by the probability any invocation of the SUBINJECTION scheme or the SUBF$_2$ scheme returns an incorrect answer. The soundness errors of both the SUBINJECTION scheme and the SUBF$_2$ scheme can be made smaller than $\frac{1}{r^c}$ for any constant $c > 0$, and therefore a union bound over all $t = O(\log_{c_v} \ell)$ invocations of each protocol implies that with high probability, no invocation of either scheme returns an incorrect answer.

5.4 Implications of the Online Scheme for Frequency Moments

Our online scheme for F_k in Theorem 5.1 has a number of important consequences.

Inner Product and Hamming Distance. Chakrabarti *et al.* [7] point out that computing inner products and Hamming Distance can be directly reduced to (exact) computation of the second Frequency Moment F_2, and so Theorems 4.5 and 5.1 immediately yield schemes for these problems of identical cost.

An improved scheme for ϕ-HEAVYHITTERS. We can use Lemma 5.3 to yield an online scheme for the ϕ-HEAVYHITTERS problem.

Corollary 5.6. *For all c_a, c_v such that $c_a c_v \ge m \log n$, there is an online $(c_a \log n \cdot \log_{c_v}(m) + \phi^{-1} \log n, c_v \log n \log_{c_v}(m))$-scheme for solving ϕ-HEAVYHITTERS in the strict turnstile update model.*

---

[2]For example, V can add one to the corresponding entry of $z^{(j)}$ for each item that is marked as an exception. This will cause $z^{(j)}$ to count the number of exceptions in each bucket, rather than indicate them, but this does not affect the correctness.

Corollary 5.6 follows from a similar analysis to Corollary 3.6. [7, Theorem 6.1] describes how to reduce ϕ-HEAVYHITTERS to demonstrating the frequencies of $O(\phi^{-1})$ items in a derived stream. Moreover, the derived stream has sparsity $O(m \log n)$ if the original stream has sparsity m. We use the MULTIINDEX scheme of Lemma 5.3 to verify these claimed frequencies.

Frequency-based functions. Chakrabarti *et al.* [7, Theorem 4.5] also explain how to extend the sum-check scheme of Proposition 4.1 to efficiently compute arbitrary *frequency-based functions*, which are functions of the form $F(\mathbf{x}) = \sum_{i \in [n]} g(f_i(\mathbf{x}))$ for an arbitrary $g : (-[N] \cup [N]) \to \mathbb{Z}$. A similar but more involved extension applies in our setting, by replacing the dense F_k scheme implied by Proposition 4.1 with the dense frequency-based functions scheme of [7, Theorem 4.5]. We spell out the details below, restricting ourselves to the prescient case for brevity; an online scheme with essentially identical costs follows by using the ideas underlying Theorem 5.1.

Corollary 5.7. *Let $F(\mathbf{x}) = \sum_{i \in [n]} g(f_i(\mathbf{x}))$ be a frequency-based function. Then there is a prescient $(N^{3/4} \log n, N^{3/4} \log n)$-scheme for computing $F(\mathbf{x})$ in the strict unit-update turnstile model. This scheme satisfies perfect completeness.*

Proof. We use a natural modification of the frequency-based functions scheme of [7, Theorem 4.5]. P specifies a hash function h at the start of the stream mapping the universe $[n]$ into $[N^{5/4}]$; P chooses h to be injective on the set of items that have non-zero frequency at the end of the stream. Using the perfect hash functions of Fredman and Komlós [15], h can be represented with $O(N^2/r \log n) = O(N^{3/4} \log n)$ bits. V stores h explicitly. After the stream is observed, P and V run the ϕ-HEAVYHITTERS scheme of Corollary 5.6, with $\phi = N^{-1/4}$. Using the fact that $\sum_i f_i < N$, by setting the parameters of Corollary 5.6 appropriately we can ensure that this part of the scheme requires annotation length $O(N^{3/4} \log n)$ and has space cost $O(N^{3/4} \log n)$. This scheme also allows V to determine the exact frequencies of the items in H, allowing V to compute $\text{cont}(H) := \sum_{i \in H} g(f_i(\mathbf{x}))$, which gives the contribution of the items in H to the output $F(\mathbf{x})$. Moreover, whenever V learns the frequency f_i of an item in $i \in H$, V treats this as a deletion of f_i occurrences of item i, thereby obtaining a derived stream $\mathbf{z}$ in which all frequencies have absolute value at most $N^{1/4}$.

P and V now run the *polynomial-agreement* scheme that was first presented in [9, Theorem 4.6] on the "mapped-down" input $h(\mathbf{z})$ over the universe $[N^{5/4}]$. For any $c_a c_v \geq r$, the polynomial agreement scheme can achieve cost $(F_{\max}(\mathbf{z}) c_a \log n, c_v \log n)$, where $F_{\max}(\mathbf{z})$ denotes $\max_i |f_i(\mathbf{z})|$, the largest frequency in absolute value of any item. Setting $c_v = N^{3/4}$ and $c_a = N^{1/4}$, we obtain a prescient $(N^{3/4} \log n, N^{3/4} \log n)$-scheme as claimed. V computes the final answer as $F(\mathbf{x}) = \text{cont}(H) + F(h(\mathbf{z})) - |H| g(0)$.

The final issue is that V needs to verify that h is actually injective over the items that appear in $\mathbf{x}$. V can accomplish this using the INJECTION scheme of Lemma 4.7. This does not affect the asymptotic costs of our scheme, as the INJECTION scheme can support annotation cost $c_a \log r$ and space cost $c_v \log r$ for any $c_a c_v = \Omega(N^{5/4})$. □

Finally, we provide one additional corollary, which describes a protocol that will be useful in the next section when building graph schemes.

Theorem 5.8. *Let $X, Y \subseteq [n]$ be sets with $|X| \leq |Y| \leq m$. Then given a stream in the strict turnstile update model with elements of X and Y arbitrarily interleaved, there is an online $(mc_v^{-1/2} \cdot \log(n) \cdot \log_{c_v}(m), c_v \cdot \log(n) \cdot \log_{c_v}(m))$-scheme for determining whether $X \subseteq Y$ for any $c_v > 1$.*

Proof. If $X \nsubseteq Y$, P can specify an $x \in X \setminus Y$ and prove that x is indeed in X and not Y with two point queries using the scheme of Theorem 3.2. For the other case, Chakrabarti *et al.* show how to directly reduce the case $X \subseteq Y$ to computation of frequency moments [7]. The claimed costs follow from Theorem 5.1. □

Table 4 provides a comparison of schemes for the SUBSET problem in the dense and sparse cases.

Scheme Costs	Completeness	Online/Prescient	Source
$(|X|\log n, \log n)$	Perfect	Prescient	[7]
$(c_a \log n, c_v \log n): c_a c_v \geq n$	Perfect	Online	[7]
$(m \log n, \log n)$	Perfect	Online	[7]
$(mc_v^{-1/2} \log_{c_v}(m) \log n, c_v \log n \log_{c_v} m): c_v > 1$	Imperfect	Online	Theorem 5.8

Table 4: Comparison of our SUBSET scheme to prior work. Ours is the first online scheme to achieve annotation length and space usage that are both sublinear in m for $m \ll \sqrt{n}$, and strictly improves over the online MA communication cost of prior protocols whenever $m = o(n)$.

6 Graph Problems

We now describe some applications of the techniques developed above to graph problems. The main purpose of this section is to demonstrate that the techniques developed within the F_k and m-DISJ schemes are broadly applicable to a range of settings.

We begin with several non-trivial graph schemes that are direct consequences of the Subset scheme of Theorem 5.8. Recall that our definition of a scheme for a function F requires a convincing proof of the value of $F(\mathbf{x})$ *for all values* $F(\mathbf{x})$. This is stricter than the traditional definition of interactive proofs for decision problems, which just require that if $F(\mathbf{x}) = 1$ then there is some prover that will cause the verifier to accept with high probability, and if $F(\mathbf{x}) = 0$ there is no such prover. Here, we consider a relaxed definition of schemes that is in the spirit of the traditional definition. We require only that a scheme $\mathcal{A} = (\mathfrak{h}, V)$ satisfy:

1. For all $\mathbf{x}$ s.t. $F(\mathbf{x}) = 1$, we have $\Pr_{r_P, r_V}[\text{out}^V(\mathbf{x}^{\mathfrak{h}, r_P}, r_V) \neq 1] \leq 1/3$.

2. For all $\mathbf{x}$ s.t. $F(\mathbf{x}) = 0$, $\mathfrak{h}' = (\mathfrak{h}'_1, \mathfrak{h}'_2, \ldots, \mathfrak{h}'_N) \in (\{0,1\}^*)^N$, we have $\Pr_{r_V}[\text{out}^V(\mathbf{x}^{\mathfrak{h}'}, r_V) = 1] \leq 1/3$.

Theorem 6.1. *Under the above relaxed definition of a scheme, each of the problems* PERFECT-MATCHING, CONNECTIVITY, *and* NON-BIPARTITENESS *has an* $(n \log n + mc_v^{-1/2} \log n \log_{c_v} m, c_v \log n \log_{c_v} m)$-*scheme on graphs with n vertices and m edges for all $c_v > 1$. All three schemes work in the strict turnstile update model and improve over prior work if $c_v = \omega(\log^2 m)$ and $c_v = o(m)$.*

Proof. In the case of perfect matching, the prover can prove a perfect matching exists by sending a matching $\mathcal{M}$, which requires $n \log n$ bits of annotation. In order to prove $\mathcal{M}$ is a valid perfect matching, P needs to prove that every node appears in exactly one edge of $\mathcal{M}$, and that $\mathcal{M} \subseteq E$, where E is the set of edges appearing in the stream. V can check the first condition by comparing a fingerprint of the nodes in $\mathcal{M}$ to a fingerprint of the set $\{1, \ldots, n\}$. V can check that $\mathcal{M} \subseteq E$ using Theorem 5.8.

In the case of connectivity, the prover demonstrates the graph is connected by specifying a spanning tree T. V needs to check T is spanning, which can be done as in [7, Theorem 7.7], and needs to check that $T \subseteq E$, which can be done using Theorem 5.8.

In the case of non-bipartiteness, P demonstrates an odd cycle C. V needs to check C is a cycle, C has an odd number of edges, and that $C \subseteq E$. The first condition can be checked by requiring P to play the edges of C in the natural order. The second condition can be checked by counting. The third condition can be checked using Theorem 5.8. $\square$

Counting Triangles. Returning to our strict definition of a scheme, we give an online scheme for counting the number of triangles in a graph.

Theorem 6.2. *For any $c_v > 1$, there is an online $(c_a \log n \log m, c_v \log n \log m)$-scheme, with imperfect completeness, for counting the number of triangles in a graph on n nodes and m edges, where $c_a = mnc_v^{-1/2}$. The scheme is valid in the strict turnstile update model.*

Scheme Costs	Completeness	Online/Prescient	Source
$(c_a \log n, c_v \log n)$: $c_a c_v \geq n^3$	Perfect	Online	[7]
$(n^2 \log n, \log n)$	Perfect	Online	[7]
$(c_a \log^2 n, c_v \log^2 n)$: $c_a = mn c_v^{-1/2}$	Imperfect	Online	Theorem 6.2

Table 5: Comparison of prior work to our scheme for counting the number of triangles in a graph with n nodes and m edges. For concreteness, notice that by setting $c_v = n$, Theorem 6.2 achieves a $(mn^{1/2} \log^2 n, n \log^2 n)$-scheme, which improves over prior work as long as $m \ll n^{3/2}$.

Proof. Chakrabarti *et al.* [7, Theorem 7.4] show how to reduce counting the number of triangles in a graph to computing the first three frequency moments of a derived stream. The derived stream has sparsity $m(n-2)$. Using the online scheme of Theorem 5.1 to compute the relevant frequency moments of the derived stream yields the claimed bounds. $\square$

The scheme of Theorem 6.2 should be compared to the $(n^2, \log n)$-scheme from [7, Theorem 7.2] based on matrix multiplication, referenced in Row 2 of Table 5 and the (h, v)-scheme for any $c_a c_v \geq n^3$ from [7, Theorem 7.3], referenced in Row 1 of Table 5. To compare to the former, notice that Theorem 6.2 yields a $(c_a \log^2 n, c_v \log^2 n)$-scheme with $c_a < n^2$ as long as $m < n\sqrt{c_v}$. To compare to the latter, note that in our new scheme, $c_a c_v = mn c_v^{1/2}$, which is less than n^3 as long as $c_v^{1/2} < \frac{n^2}{m}$. In particular, if we set $c_v = n$, then Theorem 6.2 improves over both old schemes as long as $m < n^{3/2}$.

Unfortunately, Theorem 6.2 does not yield a non-trivial MA-protocol for showing no triangle exists. Indeed, equalizing annotation length and space usage in our new protocol occurs by setting both quantities to $(mn)^{2/3}$. But $\Omega\left((mn)^{2/3}\right) < m$ only when $m > n^2$, which is to say that the MA communication complexity of this protocol is always larger than m, a cost that can be achieved by the trivial MA protocol where Merlin is ignored and Alice just sends her whole input to Bob. That is, the interest in the new protocol is that it can lower the space usage of V to less than m without drastically blowing up the message length of P to n^2 as in the matrix-multiplication based protocol from [7].

7 Non-strict Turnstile Update Model

All schemes in Sections 4 and 5 work in the strict turnstile update model. The reason these schemes require this update model is that they use the INJECTION and SUBINJECTION schemes of Lemmata 4.7 and 5.5 as sub-routines, and these sub-routines assume the strict turnstile update model.

In this section, we consider two ways to circumvent this issue. To focus the discussion, we concentrate on the online F_k protocol of Theorem 5.1.

7.1 An Online Scheme

One simple method for handling streams in the non-strict turnstile update model is the following. We use the scheme of Theorem 5.1, but within the SUBINJECTION sub-routine, we treat deletions of items in the input stream as *insertions* of items into the derived stream of (x_i, b_i, δ_i) updates. This ensures that the INJECTION and SUBINJECTION schemes correctly output 1 if the derived stream is a subinjection (and the remainder of the scheme computes the correct answer on the original stream). However it increases the expected number of collisions under the universe-reduction mappings h_i, from $m \cdot |L_{i-1}|/r$ to $M \cdot |L_{i-1}|/r$. The result is that we achieve the same costs as Theorem 5.1, except the costs depend on to the stream footprint M rather than the stream sparsity m (see Section 2.4).

Corollary 7.1. *For any $c_v > 1$, there is a $(k^2 M c_v^{-1/2} \cdot \log(n) \cdot \log_{c_v}(M), k c_v \cdot \log(n) \cdot \log_{c_v}(M))$ online scheme for F_k in the non-strict turnstile update model over a stream with footprint M over a universe of size n.*

7.2 An Online AMA Scheme

In this section, we describe an AMA scheme for the INJECTION problem that works in the non-strict turnstile stream update model i.e., the input may define a frequency vector where some elements end with negative frequency. The scheme for INJECTION of Lemma 4.7 breaks down here, since there may be some cases where the checks performed by the protocol indicate that a bucket is pure, when this is not the case: cancellations of item weights in the bucket may give the appearance of purity. To address this, we use public randomness, thereby yielding an AMA scheme. In essence, the verifier asks the prover to demonstrate the purity of each of the r buckets via fingerprints of the bucket contents. However, if we allow the prover to choose the fingerprint function, P could pick a function which leads to false conclusions. Instead, V chooses the fingerprinting function using public randomness. The players then execute a new INJECTION protocol using the data remapped under the fingerprint function, which is intended to convince V of the purity of the buckets. This then allows us to construct protocols with costs that depend on the stream sparsity m rather than the footprint M as in Corollary 7.1.

In detail, the new AMA scheme proceeds as follows. Consider the INJECTION problem as defined in Definition 4.6, but generalized to allow items with arbitrary integer counts. Consider again a bucket b, and for $1 \leq j \leq \log n$ define $b^{j=\ell}$ to be the frequency vector of the subset of stream updates (x_k, b, δ_k) placing items into bucket b, subject to the restriction that the j'th bit of x_k is equal to ℓ. We observe the following property: if bucket b is pure, then one of $b^{j=0}$ and $b^{j=1}$ must be the zero vector $\mathbf{0}$, for each j. Moreover, if b is not pure, then there exists a j such that both $b^{j=0}$ and $b^{j=1}$ are not the zero vector.

A natural way to compactly test whether these vectors are equal to zero (probabilistically) is to use fingerprinting (discussed in Section 2.4). The verifier V could do this unaided for a single bucket, but we wish to run this test in parallel for r buckets. At a high level, we achieve this as follows. Given a stream of updates (x_k, b, δ_k), we define two vectors z and o of length $r \log n$, such that each coordinate of z and o corresponds to a (bucket, coordinate) pair $(b, j) \in [r] \times [\log n]$. In more detail, we will define z and o such that for each bucket b and coordinate $j \in [\log n]$, the (b, j)th entry of z is a fingerprint of the vector $b^{j=0}$, and the (b, j)th entry of o is a fingerprint of the vector $b^{j=1}$.

We choose the fingerprinting functions to satisfy two properties.

1. The fingerprint of the all-zeros vector $\mathbf{0}$ is always 0. This ensures that if all buckets are pure, then the inner product of z and o is 0, as $z_{b,j} \cdot o_{b,j}$ is 0 for all pairs $(b, j) \in [r] \times [\log n]$.

2. If there is an impure bucket, then the inner product of z and o will be non-zero with high probability over the choice of fingerprint functions.

Therefore, in order to determine whether the stream defines an injection, it suffices to compute $\sum_{(b,j) \in [r] \times [\log n]} z_{b,j} \cdot o_{b,j}$, which can be computed using Proposition 4.1 with annotation length $c_a \log n$ and space cost $c_v \log n$ for any $c_a \cdot c_v \geq r \log n$.

The idea allowing us to achieve the second property is as follows. If bucket b is impure, then there is at least one coordinate $j \in [\log n]$ such that $b^{j=0}$ and $b^{j=1}$ are both not equal to the all-zeros vector $\mathbf{0}$. By basic properties of fingerprints, this ensures that both $z_{b,j}$ and $o_{b,j}$ are non-zero with high probability over the choice of fingerprint functions. Moreover, we choose the fingerprinting functions in such a way that non-zero terms in the sum $\sum_{(b,j) \in [r] \times [\log n]} z_{b,j} \cdot o_{b,j}$ are unlikely to "cancel out" to zero.

Consequently, we can state an analog of Lemma 4.7.

Lemma 7.2. *For any $c_a c_v \geq r \log n$, there is an online $(c_a \log n, c_v \log n)$-scheme for determining whether a stream in the non-strict turnstile model is an injection.*

Proof. Let $\mathbb{F}_q$ be a finite field of size $q = \text{poly}(n)$, where the subsequent analysis determines the required magnitude of q. V uses public randomness to choose two field elements α, and β uniformly at random from $\mathbb{F}_q$. For each bucket $b \in [r]$, and each coordinate $j \in [\log n]$, we define two "fingerprinting" functions $g_{b,j,\alpha}$ and $g_{b,j,\beta}$ mapping an n-dimensional frequency vector $\mathbb{F}$ as follows:

$$g_{b,j,\alpha}(\mathbf{x}) = \alpha^{n(b \cdot \log n + j)} \sum_{\ell \in [n]} \mathbf{x}_\ell \alpha^\ell,$$

and

$$g_{b,j,\beta}(\mathbf{x}) = \beta^{n(b \cdot \log n + j)} \sum_{\ell \in [n]} \mathbf{x}_\ell \beta^\ell,$$

where each entry $\mathbf{x}_\ell$ of $\mathbf{x}$ is treated as an element of $\mathbb{F}$ in the natural manner.

We now (conceptually) construct two vectors z and o of dimension $r \log n$, where for each $(b,j) \in [r] \times [\log n]$, $z_{b,j} = g_{b,j,\alpha}(b^{j=0})$ and $o_{b,j} = g_{b,j}(b_i^{j=1})$. That is, the (b,j)th entry of z equals the fingerprint of the frequency vector of items mapping to bucket b with a 0 in the jth bit of their binary representation. Observe that $g_{b,j,\alpha}(\mathbf{0}) = g_{b,j,\beta}(\mathbf{0}) = 0$ for all $(b,j) \in [r] \times [\log n]$, as required by Property 1 above.

We now show that Property 2 holds, i.e. if there is an impure bucket, then the inner product of z and o will be non-zero with high probability over the choice of α and β. In the following, for an item $\ell \in [n]$ and bucket $b \in [r]$, we let $f_\ell(b)$ denote the frequency with which item ℓ is mapped to bucket b, and we let ℓ_j denote the j'th bit in the binary representation of ℓ. We can write the inner product of z and o as

$$\sum_{(b,j)\in[r]\times[\log n]} g_{b,j,\alpha}(b^{j=0})g_{b,j,\beta}(b^{j=1})$$

$$= \sum_{(b,j)\in[r]\times[\log n]} \alpha^{n(b\cdot\log n+j)}\beta^{n(b\cdot\log n+j)} \left(\sum_{\ell\in[n],\ell_j=0} f_\ell(b)\alpha^\ell \right) \left(\sum_{\ell\in[n],\ell_j=1} f_\ell(b)\beta^\ell \right)$$

$$= \sum_{(b,j)\in[r]\times[\log n]} \alpha^{n(b\cdot\log n+j)}\beta^{n(b\cdot\log n+j)} \sum_{(\ell,\ell'):\ell_j=0,\ell'_j=1} f_\ell(b)f_{\ell'}(b)\alpha^\ell\beta^{\ell'}$$

We therefore see that the inner product of z and o is a polynomial in α and β of total degree $n^2 r \log n$ in each variable. Moreover, the coefficient of the term $\alpha^{n(b\cdot\log n+j)+\ell}\beta^{n(b\cdot\log n+j)+\ell'}$ is precisely $f_\ell(b) \cdot f_{\ell'}(b)$ if $\ell_j = 0$ and $\ell'_j = 1$, and is 0 otherwise.

Recall that if bucket b is not pure, then there is at least one coordinate $j \in [\log n]$, and items $\ell, \ell' \in [n]$ with $\ell_j = 0$ and $\ell'_j = 1$, such that $f_\ell(b) \neq 0$ and $f_{\ell'}(b) \neq 0$. The above analysis implies that $z \cdot o$ is a *non-zero* polynomial in α and β, as the coefficient of $\alpha^{n(b\cdot\log n+j)+\ell}\beta^{n(b\cdot\log n+j)+\ell'}$ is non-zero. Hence, by the Schwartz-Zippel lemma, the probability over a random choice of α and β that $z \cdot o = 0$ is at most $n^2 r \log n / q$. Setting q to be polynomial in n, there is only negligible probability (over the choice of α and β) that $z \cdot o$ is zero if the stream is not an injection.

Finally, notice that the verifier can apply the scheme of Proposition 4.1 to compute $\sum_{(b,j)\in[r]\times[\log n]} z_{b,j} \cdot o_{b,j}$, as each stream update (x_k, b, δ_k) can be treated as $\log n$ updates to the vectors z and o. For example, if the jth bit of x_k is 0, then update (x_k, b, δ_k) causes $z_{b,j}$ to be incremented by $\delta_k \cdot \alpha^{n(b\cdot\log n+j)+x_k}$. $\qquad\square$

Applications. We can apply this online scheme to compute Frequency Moments (and Inner Product, Hamming Distance, Heavy Hitters etc.) over sparse data in the non-strict turnstile update model. The costs of the resulting online AMA scheme are similar to the costs of the online schemes for the same problems developed in previous sections. The only difference is that we have scaled m up by a $\log n$ factor, to account for the fact that within the new AMA sub-scheme for INJECTION, we must run the dense protocol of

Proposition 4.1 on vectors z and o of length $r \log n$, rather than on vectors of length r as in prior sections, and substitute the bounds from Lemma 7.2. For example, the analog of Theorem 5.1 is that for any $c_v > 1$, there is a $(k^2 m c_v^{-1/2} \cdot \log^2(n) \cdot \log_{c_v}(m), k c_v \cdot \log(n) \cdot \log_{c_v}(m))$ online AMA scheme for F_k in the non-strict turnstile model.

8 Conclusion

We have presented a number of protocols in the annotated data streaming model that for the first time allows both the annotation length and the space usage of the verifier to be sublinear in the stream sparsity, rather than just the size of the data universe. Our protocols substantially improve on the applicability of prior work in natural settings where data streams are defined over very large universes, such as IP packet flows and sparse graph data.

A number of interesting questions remain for future work. The biggest open question is to determine the precise dependence on the stream sparsity in problems such as m-DISJ and frequency moments. When setting the annotation length and the space usage of the verifier to be equal, our protocols have cost roughly $m^{2/3}$, where m is the sparsity of the data stream. The best known lower bound is roughly $m^{1/2}$. We conjecture that our upper bound is tight up to logarithmic factors, but proving any Merlin-Arthur communication lower bound larger than $m^{1/2}$ will require new lower bound techniques in communication complexity. Another interesting open question is to give improved protocols for multiplying an $n \times n$ matrix A by a vector x, when A is sparse (i.e., has $o(n^2)$ non-zero entries), but x may be dense. Achieving this would yield improved protocols for proving disconnectedness, bipartiteness, or the non-existence of a perfect matching in a bipartite graph. Currently we do not know of any protocols for these problems that leverage graph sparsity in any way.

References

[1] S. Aaronson. QMA/qpoly ⊆ PSPACE/poly: De-Merlinizing Quantum Protocols. In *CCC*, pages 261–273, 2006.

[2] S. Aaronson and A. Wigderson. Algebrization: a new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1:1, pages 1–54, 2009. Preliminary version appeared in *STOC* 2008.

[3] L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity. In *FOCS*, pages 337–347, 1986.

[4] E. Blais, J. Brody, and K. Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21:311–358, 2012.

[5] J. Brody, A. Chakrabarti, and R. Kondapally. Certifying equality with limited interaction. Technical Report TR12-153, ECCC, 2012.

[6] H. Buhrman, D. García-Soriano, A. Matsliah, and R. de Wolf. The non-adaptive query complexity of testing k-parities. *arXiv preprint arXiv:1209.3849*, 2012.

[7] A. Chakrabarti, G. Cormode, A. McGregor, and J. Thaler. Annotations in data streams. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:22, 2012. A preliminary version of this paper by A. Chakrabarti, G. Cormode, and A. McGregor appeared in *ICALP* 2009.

[8] A. Condon. The complexity of space bounded interactive proof systems. In *Complexity Theory: Current Research*, S. Homer, U. Schöning and K. Ambos-Spies (Eds.), Cambridge University Press, pages 147–190, 1993.

[9] G. Cormode, M. Mitzenmacher, and J. Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65:2, pages 409–442, 2013. A preliminary version of this paper appeared in *ESA*, 2010.

[10] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, 2012.

[11] G. Cormode, J. Thaler, and K. Yi. Verifying computations with streaming interactive proofs. *PVLDB*, 5(1):25–36, 2011.

[12] A. Dasgupta, R. Kumar, and D. Sivakumar. Sparse and lopsided set disjointness via information theory. In *16th International workshop on Randomization*, volume 7409, pages 517–528, 2012.

[13] A. Das Sarma, R.J. Lipton, and D. Nanongkai. Best-order streaming model. *Theor. Comput. Sci.*, 412:23, pages 2544–2555 2011.

[14] D. Eppstein and M.T. Goodrich. Straggler Identification in Round-Trip Data Streams via Newton's Identities and Invertible Bloom Filters. *IEEE Trans. Knowl. Data Eng.* 23(2): 297-306, 2011.

[15] M.L. Fredman and J. Komlós. On the size of separating systems and perfect hash functions. *SIAM J. Algebra. Discr.*, 5(1):61–68, 1984.

[16] J. Håstad and A. Wigderson. The randomized communication complexity of set disjointness. *Theory of Computing*, pages 211–219, 2007.

[17] R. J. Lipton. Efficient Checking of Computations. *STACS*, pages 207–215, 1990.

[18] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[19] Y. Minsky, A. Trachtenberg, R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory* 49(9): 2213-2218, 2003.

[20] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[21] T. Gur and R. Raz Arthur-Merlin Streaming Complexity. Electronic Colloquium on Computational Complexity (ECCC). Available online at `http://eccc.hpi-web.de/report/2013/020/`, 2013.

[22] H. Klauck. Rectangle Size Bounds and Threshold Covers in Communication Complexity. In *CCC*, pages 118–134, 2003.

[23] H. Klauck. On Arthur Merlin Games in Communication Complexity. In *CCC*, pages 189–199, 2011.

[24] H. Klauck, and V. Prakash Streaming Computations With a Loquacious Prover. In ITCS, pages 305–320, 2013.

[25] M. Sağlam and G. Tardos. On the communication complexity of sparse set disjointness. Manuscript, privately communicated, 2012.

[26] A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[27] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991.

A An Online AMA Lower Bound for $(m, 2^{\sqrt{m}})$-Sparse INDEX

We prove that the online $\tilde{O}(\sqrt{m})$ protocol for the $(m, 2^{\sqrt{m}})$-Sparse INDEX problem is essentially optimal. Our lower bound follows from a natural variant of the reduction in Theorem 3.9. That is, we turn an online AMA protocol for the $(m, 2^{\sqrt{m}})$-Sparse INDEX Problem into an online MAMA protocol for the dense INDEX Problem. We then invoke a lower bound on the online MAMA communication complexity of INDEX Problem due to Klauck and Prakash [24].[3]

Theorem A.1. *The online AMA protocol complexity of the $(m, 2^{\sqrt{m}})$-Sparse* INDEX *problem is $\tilde{\Omega}(\sqrt{m})$.*

Proof. Let $n = 2^{\sqrt{m}}$. Assume we have an online AMA communication protocol $\mathcal{P}$ for (m,n)-sparse INDEX with $\mathrm{hcost}(\mathcal{P}) = \Omega(\sqrt{m})$. We describe how to use this protocol for the sparse INDEX problem to design one for the dense INDEX problem on vectors of length $n' = m\log(n/m) = \Omega\left(m^{3/2}\right)$.

Let $k = \log(n/m)$. As in the proof of Theorem 3.9, given an input x to the dense INDEX problem, Alice partitions x into n'/k blocks of length k, and constructs a vector y of Hamming weight n'/k over a universe of size $(n'/k) \cdot 2^k$ as follows. She replaces each block B_i with a 1-sparse vector $v_i \in \{0,1\}^{2^k}$, where each entry of v_i corresponds to one of the 2^k possible values of block B_i. That is, if block B_i of x equals the binary representation of the number $j \in [2^k]$, then Alice replaces block B_i with the vector $e_j \in \{0,1\}^{2^k}$, where e_j denotes the vector with a 1 in coordinate j and 0s elsewhere.

Thus, Alice now has an $n'/k = m$-sparse derived input y over a universe of size $(n'/k) \cdot 2^k = n$. Merlin looks at Bob's input to see what is the index ι of the dense vector x that Bob is interested in. Merlin then tells Bob the index ℓ such that $\ell = 2^k(\iota - 1) + j$, where B_i is the block that ι is located in, and block B_i of Alice's input x equals the binary representation of the number $j \in [2^k]$. Notice ℓ can be specified with $\log n = O(\sqrt{m})$ bits.

Alice and Bob's now use the assumed AMA-protocol for sparse disjointness to establish whether $y_\ell = 1$. If they are convinced of this, then Bob can deduce the value of *all* the bits in block B_i of the original dense vector x, and in particular, the value of x_ι.

This yields an MAMA protocol for the dense INDEX problem on $n' = \Omega(m^{3/2})$ bits. A lower bound of Klauck and Prakash [24, Lemma 7] implies that the online MAMA complexity of this problem is $\Omega((n')^{1/3}) = \Omega(m^{1/2})$. Notice also that the total hcost of our MAMA protocol is $O(\sqrt{m} + \mathrm{hcost}(\mathcal{P})) = O(\mathrm{hcost}(\mathcal{P}))$, while the vcost is $O(\mathrm{vcost}(\mathcal{P}))$. Thus, if $\mathrm{hcost}(\mathcal{P}) = \Omega(\sqrt{m})$, it must be the case that vcost is $\Omega(\sqrt{m})$ as well. This trivially implies that for any protocol $\mathcal{P}$ with hcost *less* than $\sqrt{m}$, $\mathrm{vcost}(\mathcal{P})$ must be $\Omega(\sqrt{m})$. We conclude the online AMA communication complexity of the problem is $\Omega(m^{1/2})$. This completes the proof. $\qquad\square$

---

[3] Like the lower bound of Lemma 3.8, the lower bound of Klauck and Prakash was originally proved in the communication model in which Merlin cannot send any message to Alice. However, the proof easily extends to our online MA communication model (where Merlin can send a message to Alice, but that message cannot depend on Bob's input).