Error Correction for Message Streams

Meghal Gupta
University of California Berkeley, CA, USA

Rachel Yun Zhang
MIT, Cambridge, MA, USA

Abstract

In the setting of error correcting codes, Alice wants to send a message $x \in \{0,1\}^n$ to Bob via an encoding $\mathsf{enc}(x)$ that is resilient to error. In this work, we investigate the scenario where Bob is a *low space* decoder. More precisely, he receives Alice's encoding $\mathsf{enc}(x)$ bit-by-bit and desires to compute some function $f(x)$ in low space. A generic error-correcting code does not accomplish this because decoding is a very global process and requires at least linear space. Locally decodable codes partially solve this problem as they allow Bob to learn a given bit of x in low space, but not compute a generic function f.

Our main result is an encoding and decoding procedure where Bob is still able to compute any such function f in low space when a constant fraction of the stream is corrupted. More precisely, we describe an encoding function $\mathsf{enc}(x)$ of length $\mathrm{poly}(n)$ so that for any decoder (streaming algorithm) A that on input x computes $f(x)$ in space s, there is an explicit decoder B that computes $f(x)$ in space $s \cdot \mathrm{polylog}(n)$ as long as there were not more than $\frac{1}{4} - \varepsilon$ fraction of (adversarial) errors in the input stream $\mathsf{enc}(x)$.

2012 ACM Subject Classification Mathematics of computing → Coding theory

Keywords and phrases error-correcting codes, streaming algorithms, space-efficient algorithms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2025.59

Funding *Meghal Gupta*: Supported by an NSF GRFP Fellowship.
Rachel Yun Zhang: Supported in part by DARPA under Agreement No. HR00112020023, an NSF grant CNS-2154149, and NSF Graduate Research Fellowship 2141064.

1 Introduction

Consider the following scenario: Alice streams a message denoted $x = x_1 \dots x_n$ to Bob that he receives and processes bit-by-bit. His goal is to compute some function $f(x_1 \dots x_n)$ unknown to Alice in significantly less space than necessary to store the entire stream. This scenario arises for instance in automatic control applications, where a device receives an incoming stream of data and needs to use it to make decisions.

As an example, a class of functions that Bob may wish to compute in small space from a message stream is the class of *linear functions*. If Bob's function is $f(x) = f_y(x) = x \cdot y \mod 2$ for some $y \in \{0,1\}^n$, then after receiving each bit of x, Bob adds $x_i \cdot y_i$ to a running sum. Bob only needs to track i and the running sum modulo 2, which is in total $\log n + 1$ bits of space.

However, this and other small space algorithms are very rigid when it comes to errors in the stream. Corruption of even one bit of Alice's message can change the output of Bob's linear function. The same applies if Bob's function is an index function, a majority, or the result of a decision tree.

In this work, we study what happens when there is noise in the incoming stream. In particular, we ask if it's possible to convert a given algorithm that processes a message stream into one that is robust to errors in the message while still allowing the output to be computed in low space.

In the usual error correction setting, Alice could just encode her message using a generic large distance error-correcting code. Bob would be able to compute any function f in the presence of a small constant fraction of error, but he must receive and store the whole stream in order to decode, which is far too much space. Even if Alice sends a locally decodable code, which has the property that to determine a single bit of x one only needs to track a few random bits of the codeword, it is not clear how Bob can compute a function requiring all n bits without $\Omega(n)$ storage. Our question is whether there is an encoding that preserves the space complexity of Bob's original decoding algorithm while being resilient to error.

In this work, we answer the question in the affirmative. Our main result is a scheme that protects any streaming algorithm against noise in the incoming stream. More precisely, we give an encoding $\mathsf{enc}(x)$ of length $O(n^{4+\delta})$ such that any streaming algorithm running in s space on the noiseless stream can be converted to one running in $s \cdot \text{polylog}(n)$ space on the encoded stream. It is correct with high probability whenever at most $\frac{1}{4} - \varepsilon$ of the stream was corrupted. In the specific case where Bob's function is a linear function such as dot product, our encoding can be made to have length $O(n^{2+\delta})$.

1.1 The Model

We provide a formal definition of a noise resilient transformation for a message stream and processing algorithm. The transformation has two components:

- An encoding function $\mathsf{enc}(\cdot) : X \subseteq \{0,1\}^n \to \{0,1\}^m$.
- An explicit transformation B that takes as input a deterministic streaming algorithm $A : X \to \mathbb{F}_q$ and outputs a randomized streaming algorithm $B_A = B(A) : \{0,1\}^m \to \mathbb{F}_q$.

The algorithm is said to be α-error resilient if whenever $\Delta(z, \mathsf{enc}(x)) < \alpha \cdot m$, then $B_A(z)$ outputs $A(x)$ with high probability.

1.2 Our Results

Our main result is a noise-resilient conversion for deterministic streaming algorithms.

▶ **Theorem 1.** *Fix $\varepsilon, \delta > 0$. For any $X \subseteq \{0,1\}^n$, there is a noise resilient transformation consisting of an encoding $\mathsf{enc} : X \subseteq \{0,1\}^n \to \{0,1\}^m$, and a transformation of algorithms B that is $\left(\frac{1}{4} - \varepsilon\right) \cdot m$ error resilient with probability $\geq 1 - 2^{O_\varepsilon(\log^2 n)}$. Moreover, $m = O_\varepsilon(n^{4+\delta})$, and for any deterministic algorithm A on domain X that runs in space s and time t, the algorithm $B(A)$ runs in space $s \cdot O_\varepsilon\left((\log n)^{O(1/\delta)}\right)$ and time $m \cdot O_{\varepsilon,\delta}\left(1 + \frac{t}{n^2}\right)$.*

In other words, given an algorithm A that accepts a stream of length n and uses s space, we demonstrate a noise resilient algorithm $B(A)$ computing A that uses $s \cdot \text{polylog}(n)$ space. The transformation enc of the stream x from the sender is independent of A and has length $O(n^{4+\delta})$.

A priori, it is not obvious that there is *any* low-memory noise resilient transformation of the algorithm A even with an arbitrary blow-up in communication. For example, if the sender were to encode their message using an arbitrary error-correcting code, the receiver would need to store the entire message in memory in order to decode it before applying the algorithm A. Our result shows that not only does this low-memory transformation exist, but that it can be done efficiently (both with respect to communication complexity and computational complexity).

Linear Streaming Algorithms

In the above transformation, the length of the resulting stream is $O(n^{4+\delta})$ for any $\delta > 0$. In the specific case where the streaming algorithm is *linear*, we construct a scheme where the stream length is only quadratic, namely $O(n^{2+\delta})$. The key property of linear streaming algorithms that we will be leveraging is that they can be computed in pieces in any order, and later computations do not depend on the results of previous ones.

▶ **Definition 2** (Linear Streaming Algorithms). *A linear streaming algorithm $A : \{0,1\}^n \to \mathbb{F}_q$ is described by a list of functions $g_i : \{0,1\} \to \mathbb{F}_q$ for $i \in [n]$, and computes the value $A(x) = g_1(x_1) + \ldots + g_n(x_n)$ (where addition is over $\mathbb{F}_q$) by tracking the partial sum $g_1(x_1) + \ldots + g_i(x_i)$ upon receiving the i'th bit.*

Note, for example, that every linear function on codomain $\{0,1\}^n$ admits a linear streaming algorithm. We note that linear streaming algorithms capture a large class of interesting functions, including linear sketching algorithms (see for example [12, 10, 1, 37]). For linear streaming algorithms, we show the following result.

▶ **Theorem 3.** *Fix $\varepsilon, \delta > 0$. There is a function $\mathsf{enc} : X \subseteq \{0,1\}^n \to \{0,1\}^m$ where $m = O_\varepsilon(n^{2+\delta})$ and an explicit transformation B such that the following holds: For any linear streaming algorithm A that takes as input $x \in X \subseteq \{0,1\}^n$ as a stream, runs in space s and time t, and outputs $A(x)$, the algorithm $B_A = B(A)$ takes as input $z \in \{0,1\}^m$ as a stream, runs in space $s \cdot O_\varepsilon((\log n)^{O(1/\delta)})$ and time $m \cdot O_{\varepsilon,\delta}\left(1 + \frac{t}{n^2}\right)$, and satisfies that whenever $\Delta(z, \mathsf{enc}(x)) < \left(\frac{1}{4} - \varepsilon\right) \cdot m$ then $B_A(z)$ outputs $A(x)$ with probability $\geq 1 - 2^{O_\varepsilon(\log^2 n)}$.*

Randomized Algorithms

We remark that our transformations in Theorems 1 and 3 are only for deterministic algorithms. However, this easily implies the result for algorithms A that *pre-sample* all of their randomness (that is, algorithms that fix their randomness before receiving any bits of x, at which point they are deterministic for the remainder of the algorithm). Such algorithms can be viewed as sampling from a distribution over deterministic algorithms. Our transformation can then be applied to the ensuing deterministic algorithm, thus correctly computing the function A with high probability while being resilient to noise.

We remark that this notion of randomized algorithms whose only access to randomness is pre-sampled at the start of computation is quite general: randomized algorithms that sample its randomness online can often be converted to protocols where the randomness is pre-sampled, see e.g. [44].

Non-Binary Input Alphabets

Our main theorems are stated for the setting where the input stream is binary. We remark that this assumption is without loss of generality: a larger alphabet stream can be converted to a binary one by assigning each alphabet symbol to a unique binary string.

1.3 Discussion of Subsequent Work

Since this paper was first posted, the work of [29] improved upon our schemes in a few major ways.

- For the case of general deterministic streaming algorithms, they demonstrate a transformation that only requires near-quadratic encoding length.

- They show that this is essentially tight: there is no transformation that has sub-quadratic encoding length.
- And, in the case of linear streaming algorithms, they construct a transformation that has an encoding of near linear length.

This addresses most of the open questions posed by our paper.

1.4 Related Works

Streaming Algorithms

The study of streaming algorithms began with the work Morris [42] on approximate counting; a rigorous analysis was given later in [20]. Later works introduced other classic problems in streaming, including heavy hitters [9], ℓ_p approximation [2, 41, 36], and finding a nonzero entry in a vector (for turnstile algorithms) [41].

Many works, for example [22, 11, 39, 3], consider the problem of processing noisy data sets using streaming algorithms. [22] shows a memory lower bound for learning a parity function with noisy samples of random linear equations.

However, this type of noise is quite different from our treatment. In these problems, noise is inherent in the way samples are generated, whereas we are investigating noise that occurs in the communication process.

Error Correcting Codes

Our result can be viewed through the lens of low-space noise resilient one-way communication.

Noise resilient one-way communication, in other words, error correction [51, 35], is one of the most well-studied problems in computer science. One specific line of works related to our result is that of *locally decodable codes*. Locally decodable codes [52] can be viewed as a low-time and low-space version of error-correcting codes, where the goal is to learn a single bit of the original message. By constrast, for us, the decoder must be able to piece together any function of the input that is noiselessly computable in low space, with the tradeoff that the decoder accesses the entire encoding via a stream rather than via query access. Local decodable codes have been constructed in a variety of different parameter regimes, including constant query [16, 15] and near 1 rate [38]. In our work, we will use Reed-Muller codes [43, 46] that achieve polylog(n) query complexity and slightly super-linear block length.

Another line of works related to ours is that of streaming algorithms for local testing of codes [47, 40]. However, this direction was rendered moot by the recent discovery of locally testable codes with constant rate, distance, and locality [14].

Coding for Interactive Communication

The analogue of noise resilient one-way communication in the interactive setting, where Alice and Bob have inputs x, y and engage in a protocol to compute some $f(x, y)$ while being resilient to error, comprises the study of *coding for interactive communication*. Interactive coding was studied starting with the seminal works of Schulman [48, 49, 50] and continuing in a prolific sequence of followup works, including [8, 6, 4, 5, 34, 7, 13, 25, 24, 17, 28, 26, 19, 30, 31]. We refer the reader to an excellent survey by Gelles [23] for an extensive list of related work.

The recent work of [18] studies the space complexity of interactive coding schemes. Their main result is an interactive coding scheme that preserves the space complexity of the original noiseless protocol up to a $\log(n)$ factor, where n is the length of the protocol. Their result

can be viewed as the interactive analogue of ours, where their noiseless and noise resilient protocols are both interactive. We remark that their techniques are quite different than ours and do not apply to our setting, however, since their approach crucially relies on the communication of feedback from Bob to Alice.

2 Overview of Techniques

Consider the task of computing a linear function $f(x) = \sum_{i\in[n]} g_i(x_i)$. In a noiseless setting, one can compute $f(x)$ by tracking the partial sum $\sum_{1\le i\le n'} g_i(x_i)$ and updating it with $g_{n'+1}(x_{n'+1})$ upon receiving the $(n'+1)$'th bit of x.

As a first attempt to create a noise resilient version of this algorithm, one might consider the input stream $\mathsf{enc}(x) = \mathsf{LDC}(x)$, where LDC is a locally decodable code. In short, a locally decodable code is a noise resilient encoding of x such that in order to decode any specific bit of x, say x_i, the decoder simply has to read polylog(n) (randomized) bits of $\mathsf{LDC}(x)$. In a streaming setting, the decoder can record only these polylog(n) bits into memory, then run the decoding algorithm to determine x_i after all such query bits have been seen.

However, it's not clear that this local decoding property extends to arbitrary linear functions.[1] If we attempt to simultaneously decode all bits x_i from $\mathsf{LDC}(x)$ using a separate query set Q_i for each index, the issue is that we will need to track $\Omega(n)$ queries simultaneously since we cannot ensure that the local decoding for any index finishes before another: the query sets Q_i are typically randomized so that any bit of the codeword is equally likely to belong to the query set for any particular index.

A second attempt is to send $\mathsf{LDC}(x)$ n times, with the intent that Bob locally decodes x_i in the i'th $\mathsf{LDC}(x)$ and computes $f(x)$ by tracking the partial sum $\sum_{1\le i\le n'} g_i(x_i)$ at the end of the i'th chunk. Now, the only space requirements are the space required to store this partial sum and the space required to locally decode x_i in each $\mathsf{LDC}(x)$, which is polylog(n). However, the adversary can simply corrupt the first $\mathsf{LDC}(x)$ so that Bob mis-decodes x_1, thereby corrupting his final output, and ultimately, this approach is not so different from just sending the bits $x_1 \dots x_n$ in order.

What Bob would like to do is randomize which x_i he is computing from a given copy of $\mathsf{LDC}(x)$ so that the adversary does not know. This crucially uses the property that LDC's allow Bob to decode *any* x_i not known to the encoder or adversary. Then, if he computes each x_i many times and takes the majority, he will learn x_i correctly with high probability, regardless of the adversary's corruption process. Since the adversary does not know in advance which chunks Bob is computing x_i in, she cannot target corruptions towards a specific x_i. However, this runs into an issue with the amount of space Bob requires. Since he computes each x_i many times throughout the protocol and only takes the majority when he has finished these computations, he needs to track his current guess for its value through (almost) the entire protocol. When each x_i was computed only once, Bob could just add $g_i(x_i)$ to the running sum, but now he needs to track each x_i individually between the first and last time he queries it, for a total of $\Omega(n)$ bits of space.

The crux of our construction is an organized system for aggregating decoding attempts in space much smaller than storing all n attempts simultaneously.

[1] The smallest local decodable code we know of that supports local decoding to any arbitrary linear function is an extension of the Hadamard code, which has block length exponential in n. This will not be a satisfactory solution for us because (a) we would like encodings that are efficiently computable, and (b) keeping track of which bit of the stream one is receiving takes n memory.

A recursive computation

Let $r = \text{polylog}(n)$. Suppose that we already know how to compute linear functions on inputs of size n/r in space $s_{n/r}$, that is resilient to $\varepsilon_{n/r}$ fraction of errors in the stream of $M_{n/r}$ copies of $\mathsf{LDC}(x)$. We will use this to build a scheme for computing linear functions on inputs of size n. Note that the base case is given by computing linear functions on a single bit, which can be done by local decoding on a single $\mathsf{LDC}(x)$.

To compute $f(x) = \sum_{i\in[n]} g_i(x_i)$, we just need to compute $f_1(x) = \sum_{i\in[n/r]} g_i(x_i)$, $f_2(x) = \sum_{i\in[n/r+1,2n/r]} g_i(x_i)$, $\ldots$, $f_r(x) = \sum_{i\in[n-n/r+1,n]} g_i(x_i)$. Each sub-computation can individually be done with the guarantee that if $< \varepsilon_{n/r}$ fraction of the stream is corrupted, then the function is computed correctly.

Consider a stream consisting of $\ell \cdot r \cdot M_{n/r}$ copies of $\mathsf{LDC}(x)$ (you can think of $\ell = \log^2 n$). We split up this stream into ℓ chunks each consisting of r blocks of $M_{n/r}$ $\mathsf{LDC}(x)$'s. In each of the ℓ chunks, we will assign the r blocks of $M_{n/r}$ LDC's to the computation of a random permutation of $f_1(x), \ldots, f_r(x)$. Throughout the algorithm, we will keep track of all the outputs from each of the ℓ sub-computations for each of $f_1(x), \ldots, f_r(x)$. At the end, we can take a majority vote for each of the $f_j(x)$. We illustrate the recursion process in Figure 1.

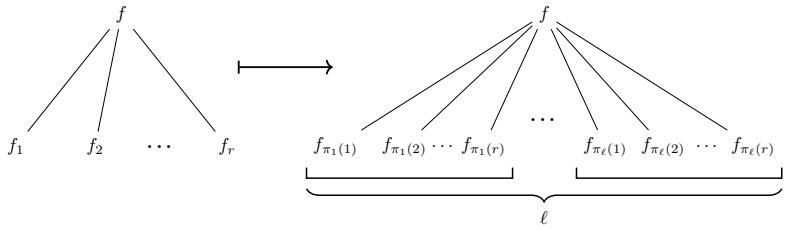

Figure 1 In our transformation, each sub-function is computed ℓ times instead of 1, and in each of the ℓ chunks, the sub-functions f_i are computed in according to a random permutation π.

Because of the way we've randomly assigned blocks to $f_1, \ldots, f_r$, the adversary cannot concentrate her attack on the computation of any specific f_i. More precisely, we can show that the amount of error present in the blocks corresponding to the computation of f_k is (roughly) proportional to the total error present. Then, as long as at least half the blocks corresponding to f_k have $< \varepsilon_{n/r}$ fraction of errors, we have that the majority vote for each of $f_1, \ldots, f_r$ will be correct. In total, this means that our algorithm for computing $f(x)$ is resilient to $\frac{\varepsilon_{n/r}}{2}$ fraction of errors.

Unfortunately, this factor of 2 loss in the error resilience in each iteration of the recursion will prove to be problematic, as we can no longer guarantee a positive error resilience when n gets large.

Decoding with confidences

In the above analysis, our worst case error distribution was that where just under one-half of the chunks have $\varepsilon_{n/r}$ error while the rest are error-free. In this case, the chunks with $\varepsilon_{n/r}$ error will produce a wrong output, while the error-free ones will produce a correct output.

However, we notice that in certain local decoding constructions (for us, we will use a Reed-Muller code), one can actually obtain an estimate of how much error there was in the codeword based on the inconsistency of the queries. In particular, in chunks where there were $\varepsilon_{n/r}$ errors, even though we will obtain a wrong answer, we can also see that there were a lot of errors in the codeword, and thereby be less sure of the obtained answer.

This motivates our definition of *local decoding with confidences*: in addition to outputting an answer, the decoding algorithm should also output a *confidence* $c \in [0, 1]$ indicating how confident it is that the answer is correct. A confidence of 0 indicates that it's not sure at all: the fraction of errors seen was near the threshold ε_1; whereas a confidence of 1 indicates that it's quite sure: either the answer is right and no errors occurred, or at least $2\varepsilon_1$ errors were required to obtain a wrong answer with confidence 1. Another way to view the confidence is as an estimate of how much error was present in the total codeword (how far the actual error is from ε_1).

Now, when we perform the recursive step to obtain a noise resilient algorithm for f from $f_1, \ldots, f_r$, in addition to recording the outputted answer from each chunk, we will also record the associated confidence. Suppose that the ℓ answers and confidences obtained for f_k were $(\hat{q}^{(1)}, \hat{c}^{(1)}), \ldots, (\hat{q}^{(\ell)}, \hat{c}^{(\ell)})$. Then, the weighted majority vote of these answers is defined to be the $\hat{q}$ with the largest total confidence. We can show that as long as the cumulative fractional error c_n is at most $c_{n/r}$, we're guaranteed that the correct answer has more the highest total confidence.

We remark that one does not have to store all ℓ pairs $(\hat{q}^{(j)}, \hat{c}^{(j)})$. Rather, we can keep track of a most likely q and associated confidence c. To update (q, c) with a new pair $(\hat{q}, \hat{c})$, one can update c by $\pm\hat{c}$ ($+\hat{c}$ if $\hat{q} = q$, and $-\hat{c}$ otherwise). If the resulting confidence c is less than 0, then it means that the answer $\hat{q}$ is supported by less overall error, so we set our most likely q to be $\hat{q}$ and flip the sign of c.

Now, we analyze the space and communication complexity of the resulting algorithm. The space required is simply $s_{n/r}$, the space required to do a size n/r sub-computation, plus the space required to store a pair (q, c) for each of $f_1, \ldots, f_r$. Overall, this means that for each layer of the recursion, we are only gaining an additive factor of $r \cdot (|q| + |c|)$ space overhead, so the total space overhead is at most polylogarithmic in n. As for the size of the resulting stream, we see that the number of $\mathsf{LDC}(x)$'s is $\approx (r \cdot \ell)^{\log_r n}$,[2] which is polynomial in n.

Computing non-linear functions

Our algorithm so far has only captured functions f that are splittable into r sub-computations whose computations do not depend on each other. However, for a general function, this may not be the case. Indeed, for many functions, the computations must be done *sequentially*, where later computations can only be done once earlier computations have finished. We may split a general function f into r sub-functions where $q_1 := f_1(x[1 : n/r], \emptyset)$ is the state of the algorithm after n/r bits of the stream, $q_2 := f_2(x[n/r + 1 : 2n/r], q_1)$ is the state of the algorithm after receiving the next n/r bits of the stream, and so on. The difference from the linear function case is that to compute f_j, we must have the correct output of f_{j-1}.

In order to handle such sequential computations, we modify our above algorithm as follows. In the recursive layer computing f from functions $f_1, \ldots, f_r$, we always compute f_j from the starting state q_{j-1} which is our current best guess for the output of f_{j-1}. This value of q_{j-1} may not always be correct, in which case the computation done for f_j will be doomed.

As with our algorithm for linear functions, after every chunk, we update the most likely state q_j and confidence c_j for the sub-function that was computed in that chunk. However, when a state q_j changes, this also means that computations done for $f_{j+1} \ldots f_n$ were based

[2] In our actual protocol, we will repeat the procedure $\log^2 n$ times and take a final majority vote, so the number of $\mathsf{LDC}(x)$'s used will be $(r \cdot \ell)^{\log_r n} \cdot \log^2 n$.

on wrong information. In the case of linear functions, the computations are independent so this does not matter, but for sequential functions, we have to discard the guesses $q_{j+1} \dots q_n$. Formally, we set these states to $\emptyset$ and set the corresponding confidences all to 0.

Because each sub-function q_j is only computed usefully when q_{j-1} is correct and also the value of q_j is frequently discarded, it is less clear that over many iterations the q_j's will converge to the correct values. Nevertheless, we prove in Section 6 that over enough iterations, the guesses q_j will eventually all be correct as long as there are not too many errors in the stream.

Outline

The rest of this paper is organized as follows. In Section 3, we start with some preliminary theorems about error-correcting codes. In Section 4, we define our notion of local decoding with confidences and prove that Reed-Muller codes satisfy this property. Then, in Sections 5 and 6, we present our noise resilient streaming algorithms for linear and general (sequential) functions, respectively.

3 Preliminaries

Notation

- The function log is in base 2 unless otherwise specified.
- The set $[n]$ denotes the integers $1 \dots n$.
- We use $x[i:j]$ to denote the i'th to j'th bits of x inclusive. We use $x(i:j]$ to refer to the $(i+1)$'th to j'th bit inclusive.
- We use e as a variable rather than the universal constant e.

3.1 Error-Correcting Codes

We begin with some results about error-correcting codes. The first is the existence of a relative distance $\frac{1}{2}$ binary code.

▶ **Theorem 4** ([33]). *For all $\epsilon > 0$, there exists an explicit error-correcting code $\mathsf{ECC}_\epsilon = \{\mathsf{ECC}_{\epsilon,n} : \{0,1\}^n \to \{0,1\}^m\}_{n \in \mathbb{N}}$ with relative distance $\frac{1}{2} - \epsilon$ and $m = O_\epsilon(n)$, and a $\mathrm{poly}_\epsilon(n)$-time decoding algorithm $\mathsf{DEC}_\epsilon : \{0,1\}^m \to \{0,1\}^n$, such that for any $x \in \{0,1\}^n$ and $w \in \{0,1\}^m$ satisfying $\Delta(\mathsf{ECC}_\varepsilon(x), w) < \left(\frac{1}{4} - \frac{\varepsilon}{4}\right) \cdot m$, it holds that $x = \mathsf{DEC}_\varepsilon(w)$.*

Our second theorem is about the efficient decoding of Reed-Solomon codes [45].

▶ **Theorem 5** (Berlekamp-Welch; see e.g. [27]). *Given a collection of tuples $(\alpha_1, v_1), \dots, (\alpha_n, v_n)$ where $\alpha_i, v_i \in \mathbb{F}_q$ and a parameter $d \leq n$, there is an algorithm running in $\mathrm{poly}(n, \log q)$ time that outputs a polynomial $g(\alpha) \in \mathbb{F}_q[\alpha]$ of degree at most $d-1$ such that $\Delta((g(\alpha_i))_{i \in [n]}, (v_i)_{i \in [n]}) < \frac{n-d+1}{2}$.*

Finally, we recall generalized minimum distance decoding, which allows us to decode concatenated codes efficiently.

▶ **Theorem 6** (GMD Decoding [21, 32]). *Given two codes, $N_{outer} : \{0,1\}^k \to \Sigma^n$ of distance $1 - \varepsilon_{outer}$ with decoding time T_{outer}, and $N_{inner} : \Sigma \to \{0,1\}^m$ of distance $\frac{1}{2} - \varepsilon_{inner}$ with decoding time D_{inner}, there is an algorithm running in time $\mathrm{poly}(D_{inner}, D_{inner})$ that on input $w \in \{0,1\}^{n \cdot m}$ outputs $x \in \{0,1\}^k$ such that $\Delta(w, N_{outer} \circ N_{inner}(x)) < \frac{(1-\varepsilon_{outer})(\frac{1}{2}-\varepsilon_{inner})}{2}$.*

4 Locally Decoding with Confidences

We now recall the definition of *locally decodable codes*. For us, besides just correctness of local decoding when the distance to a codeword is small, we require that the decoder output a *confidence* indicating how much error it witnessed (less error means higher confidence).

▶ **Theorem 7** (Locally Decodable Code). *For any $\varepsilon > 0$ and $d = d(n) = \log^{1/\delta} n$, there is a code $\mathsf{LDC} : \{0,1\}^n \to \{0,1\}^{N(n)}$ where $N(n) = O_\varepsilon(n^{1+\delta})$ that satisfies the following properties:*

- **Distance:** *For any $x \neq y \in \{0,1\}^n$, it holds that $\Delta(\mathsf{LDC}(x), \mathsf{LDC}(y)) \geq \left(\frac{1}{2} - \varepsilon\right) \cdot N$.*
- **Locally Decoding with Confidence:** *For any index $i \in [n]$ and any word $w \in \{0,1\}^N$, there exists a randomized algorithm $\mathcal{D}_i$ that reads $Q = O_\varepsilon(\log^{2/\delta} n)$ bits of w, runs in $O_\varepsilon(n^{O(1/\delta)})$ time, and outputs a bit $\hat{x}_i$ along with confidence $\mathsf{conf} \in [0,1]$, satisfying that*
  - *For any $x \in \{0,1\}^n$ such that $\Delta(w, \mathsf{LDC}(x)) = \left(\frac{1}{4} - 2\varepsilon - e\right) \cdot N$ where $e \geq 0$, it holds that*

    $$\hat{x}_i = x_i \qquad \text{and} \qquad \mathsf{conf} > e$$

    with probability at least $1 - \exp\left(-\log^2 n\right)$.
  - *For any $x \in \{0,1\}^n$ such that $\Delta(w, \mathsf{LDC}(x)) = \left(\frac{1}{4} - 2\varepsilon + e\right) \cdot N$ where $e > 0$, it holds that*

    $$\hat{x}_i = x_i \qquad \text{or} \qquad \mathsf{conf} < e$$

    with probability at least $1 - \exp\left(-\log^2 n\right)$.

  Furthermore, the confidence conf can be represented as a rational number with denominator $4Q$.

Proof. This proof is omitted for the conference version. ◀

5 Noise Resilient Streaming for Linear Algorithms

Our first result is a noise resilient conversion for linear streaming algorithms A. We begin by recalling the definition of a linear streaming algorithm.

▶ **Definition 2** (Linear Streaming Algorithms). *A linear streaming algorithm $A : \{0,1\}^n \to \mathbb{F}_q$ is described by a list of functions $g_i : \{0,1\} \to \mathbb{F}_q$ for $i \in [n]$, and computes the value $A(x) = g_1(x_1) + \ldots + g_n(x_n)$ (where addition is over $\mathbb{F}_q$) by tracking the partial sum $g_1(x_1) + \ldots + g_i(x_i)$ upon receiving the i'th bit.*

For such algorithms, we describe a noise resilient conversion that incurs quadratic blow-up in communication complexity.

▶ **Theorem 3.** *Fix $\varepsilon, \delta > 0$. There is a function $\mathsf{enc} : X \subseteq \{0,1\}^n \to \{0,1\}^m$ where $m = O_\varepsilon(n^{2+\delta})$ and an explicit transformation B such that the following holds: For any linear streaming algorithm A that takes as input $x \in X \subseteq \{0,1\}^n$ as a stream, runs in space s and time t, and outputs $A(x)$, the algorithm $B_A = B(A)$ takes as input $z \in \{0,1\}^m$ as a stream, runs in space $s \cdot O_\varepsilon((\log n)^{O(1/\delta)})$ and time $m \cdot O_{\varepsilon,\delta}\left(1 + \frac{t}{n^2}\right)$, and satisfies that whenever $\Delta(z, \mathsf{enc}(x)) < \left(\frac{1}{4} - \varepsilon\right) \cdot m$ then $B_A(z)$ outputs $A(x)$ with probability $\geq 1 - 2^{O_\varepsilon(\log^2 n)}$.*

5.1 Statement of Algorithm

Let $\varepsilon > 0$. Throughout the section let $r = \log^{1/\delta} n$ and define $\ell := \log^2 n$. We also assume for simplicity that $\log_r n$ is an integer. We will also assume that $n > \max\{\exp(\exp(8\delta/\varepsilon)), \exp(1/\varepsilon)\}$ is sufficiently large, so that also $\ell < n$.

We begin by specifying the encoding $\mathsf{enc}(x)$ that Alice sends in the stream.

▶ **Definition 8** ($\mathsf{enc}(x)$)**.** *Let $\mathsf{LDC}(x)$ be a locally decodable code with $N(n) = |\mathsf{LDC}(x)| = O_\varepsilon(n^{1+\delta})$ and query complexity $Q = O_\varepsilon(\log^{2/\delta} n)$ satisfying the guarantees of Theorem 7 for $\varepsilon/8$. Then Alice sends $\mathsf{enc}(x) := \mathsf{LDC}(x)^{\log^2 n \cdot M_n}$ where $M_n = (r \cdot \ell)^{\log_r n}$ (that is, $\mathsf{enc}(n)$ is $\log^2 n \cdot M_n$ copies of $\mathsf{LDC}(x)$). In particular, $m = |\mathsf{enc}(x)| = (r \cdot \ell)^{\log_r n} \cdot \log^2 n \cdot N(n) = O_\varepsilon(n^{2+4\delta})$.*

Throughout this section, we use the notation $A_{i,j}(x)$ to denote the quantity $g(x_{i+1}) + \ldots + g_j(x_j)$. Then, $A(x) = A_{0,n}(x)$. We define $N_{j-i} := (r \cdot \ell)^{\log_r(j-i)} \cdot N(n)$ which, as we'll see in the description below, represent the number of bits read by the algorithm to approximate $A_{i,j}(x)$.

We describe Bob's streaming algorithm $B_A := B(A)$. Before stating it formally, we provide a high-level description.

Description

Let z be the incoming stream that Bob receives. At a high level, the algorithm B_A runs by generating many guesses for $A(x)$ along with confidences indicating how likely that guess was correct (depending on how much error is witnessed while generating the guess). At the end, the guess that has the most cumulative confidence is outputted.

For i, j where $j - i$ is an integer power of r, we define the algorithm $\mathsf{estA}(i, j)$ that takes in two input two indices $i, j \in [n]$. It reads N_{j-i} many bits of z after which it outputs a guess for $A_{i,j}(x)$ along with a confidence. In particular, $\mathsf{estA}(0, n)$ outputs a guess for $A(x)$ along with a confidence. By running $\mathsf{estA}(0, n)$ many times and aggregating the guesses weighted by the confidences, we obtain a final output.

To compute $\mathsf{estA}(i, j)$, we break down the interval $(i : j]$ into r subintervals $(i_0 : i_1], (i_1 : i_2], \ldots, (i_{r-1} : i_r]$ where $i_a = i + a \cdot \frac{j-i}{r}$. We then recursively formulate guesses for each $A\left(x\left(i_{a-1} : i_a\right]\right)$ over many iterations by calling $\mathsf{estA}(i_{a-1}, i_{a)}$ each many times. The choice of a in each iteration must be randomized, so that the adversary cannot attack any single choice of a. More specifically, we will split up the N_{j-i} length input stream into ℓ chunks, each of length N_{j-i}/ℓ bits. Each chunk is split into r sections, each of size $N_{j-i}/(\ell \cdot r) = N_{(j-i)/r}$. In each chunk, we pick a random permutation $(\pi_1, \pi_2, \ldots, \pi_r)$ of $[r]$. In the a'th section of the chunk, we will compute on the subproblem corresponding to interval $(i_{\pi_a - 1} : i_{\pi_a}]$ by calling $\mathsf{estA}\left(i_{\pi_a - 1}, i_{\pi_a}\right)$.

Whenever a recursive call to $\mathsf{estA}\left(i_{a-1}, i_a\right)$ is completed, outputting $(\hat{q}, \hat{c})$, then: if $q_a = \hat{q}$ then the confidence c_a is increased by $\hat{c}$, and if $q_a \neq \hat{q}$ then the confidence c_a is decreased by $\hat{c}$. If the confidence c_a becomes negative, then we are more sure that the correct state is $\hat{q}$ rather than q_a: then q_a is replaced by $\hat{q}$ and the confidence is negated (so that it's positive).

5.2 Proof of Theorem 3

We begin by proving the stated claims about the communication, time, and space complexities, and then proceed to prove correctness.

Algorithm 1 Bob's Noise Resilient Algorithm B_A.

input $n \in \mathbb{N}$ and stream $z \in \{0,1\}^m$.
function $\mathsf{estA}(i,j)$ ▷compute the state of A starting at state q between steps i and j
  if $j = i+1$ **then**
    Read the next $|\mathsf{LDC}(x)|$ bits of the stream y.
    Using Theorem 7 compute guess $\hat{b}$ for $x[j]$ and confidence $\hat{c}$ in $O_\varepsilon\left((\log n)^{O(1/\delta)}\right)$ bits of space.
    return $g_i(\hat{b}), c$.
  else
    Let $i_0 = i, i_1 = i + \frac{j-i}{r}, i_2 = i + \frac{2(j-i)}{r}, \ldots, i_r = j$.
    Initialize a list of pairs $(q_1, c_1), \ldots (q_r, c_r)$ each to $(\emptyset, 0)$. ▷cumulative best guesses and confidences
    for ℓ iterations **do**
      Let $\pi_1 \ldots \pi_r$ be a random permutation of $[r]$.
      for $a \in [r]$ **do**
        Compute $(\hat{q}, \hat{c}) \leftarrow \mathsf{estA}(i_{\pi_a - 1}, i_{\pi_a})$.
        if $\hat{q} = q_{\pi_a}$ **then** ▷update the confidence on $\mathsf{estA}(i_{\pi_a}, i_{\pi_j + 1}, q_{\pi_a - 1})$
          $c_{\pi_a} \leftarrow c_{\pi_a} + \hat{c}$
        else
          $c_{\pi_a} \leftarrow c_{\pi_a} - \hat{c}$
          if $c_{\pi_a} < 0$ **then** ▷if the guess changes, flip its confidence
            $q_{\pi_a} \leftarrow \hat{q}$ and $c_{\pi_a} \leftarrow -c_{\pi_a}$.
    return $q_1 + \ldots + q_r, \min(c_1 \ldots c_r)/\ell$

Initialize a pair (q, c) to $(\emptyset, 0)$.
for $\log^2 n$ iterations **do** ▷amplification step
  Let $(\hat{q}, \hat{c}) \leftarrow \mathsf{estA}(0, n)$
  if $\hat{q} = q$ **then**
    $c \leftarrow c + \hat{c}$
  else
    $c \leftarrow c - \hat{c}$
    if $c < 0$ **then**
      $q \leftarrow \hat{q}$ and $c \leftarrow -c$.
output q. ▷output $A(x)$

5.2.1 Algorithmic Complexity

The proofs of communication, time, and space complexities are omitted for the conference version.

5.2.2 Correctness

We now show correctness. Formally, correctness is shown by the following lemma.

▶ **Lemma 9.** *When at most* $\frac{1}{4} - \varepsilon$ *fraction of the stream* $\mathsf{enc}(x)$ *is corrupted, Algorithm 1 outputs* $A(x)$ *with probability* $\geq 1 - \exp\left(-\varepsilon^2 \log^2 n/32\right)$.

We prove the following statement which will easily imply Lemma 9. For a given i, j and associated string that is read by the algorithm in the computation, let the random variable $(q(i,j), c(i,j))$ denote the state and confidence after the computation $\mathsf{estA}(i,j)$. We

define the *signed confidence*, denoted $c^{\pm}(i,j)$, to be defined as $+c(i,j)$ if $q(i,j) = A_{i,j}(x)$ and $-c(i,j)$ otherwise. For intuition, the more positive $c^{\pm}(i,j)$ is, the more correct with higher confidence the output $(q(i,j), c(i,j))$ is, whereas the more negative $c_{\pm}(i,j)$ is, the more $q(i,j)$ is incorrect with high confidence. So, $c^{\pm}(i,j)$ gives us a scaled measure of how correct the output of $\mathsf{estA}(i,j)$ is.

▶ **Lemma 10.** *For any $0 \leq i < j \leq n$ and $e \in \mathbb{R}$, given $\leq \frac{1}{4} - \frac{\varepsilon}{4} - \frac{\varepsilon}{4} \cdot \frac{\log(j-i)}{\log n} - e$ fraction of corruptions in the bits read by the computation of $\mathsf{estA}(i,j)$, we have $\mathbb{E}[c^{\pm}(i,j)] > e$.*

Proof. The proof of this lemma is omitted for the conference version. ◀

Proof of Lemma 9. In the special case where $i = 0, j = n$ and there are $\leq \frac{1}{4} - \frac{\varepsilon}{2} - e < \frac{1}{4} - \frac{\varepsilon}{4} - \frac{\varepsilon}{4} \cdot \frac{\log(j-i)}{r} - e$ errors (where the inequality follows because $\log n < \varepsilon \cdot r$), by Lemma 10 we have that $\mathbb{E}[c^{\pm}(0,n)] > e$.

Then, the final amplification step of the protocol computes the pair $(q(0,n), c(0,n))$ for $\log^2 n$ times, where in each chunk i we denote the fraction of error to be $\left(\frac{1}{4} - \frac{\varepsilon}{2} - e_i\right)$, where $\sum_i e_i \geq \frac{\varepsilon \log^2 n}{2}$ since we assumed the total error was $\leq \frac{1}{4} - \varepsilon$. Also, the protocol outputs $A(x)$ if and only if $\sum_i c_i^{\pm}(0,n)$ (denoting the value of $c^{\pm}(0,n)$ in the i'th chunk) is positive. By Azuma's inequality,

$$Pr\left[\sum_i c_i^{\pm}(0,n) > 0\right] \geq Pr\left[\sum_i \left(c_i^{\pm}(0,n) - e_i\right) > -\frac{\varepsilon \log^2 n}{2}\right] \geq 1 - \exp\left(-\frac{\varepsilon^2 \log^2 n}{32}\right).$$ ◀

This concludes the proof of Theorem 3.

6 Noise Resilient Streaming for General Algorithms

We now consider general streaming algorithms, whose computational may be sequential in nature. Our main result is a noise resilient conversion for deterministic streaming algorithms A. Compared to our scheme for linear algorithms from Section 5, the length of our encoding is larger: $n^{4+\delta}$ compared to $n^{2+\delta}$.

▶ **Theorem 1.** *Fix $\varepsilon, \delta > 0$. For any $X \subseteq \{0,1\}^n$, there is a noise resilient transformation consisting of an encoding $\mathsf{enc} : X \subseteq \{0,1\}^n \to \{0,1\}^m$, and a transformation of algorithms B that is $\left(\frac{1}{4} - \varepsilon\right) \cdot m$ error resilient with probability $\geq 1 - 2^{O_\varepsilon(\log^2 n)}$. Moreover, $m = O_\varepsilon(n^{4+\delta})$, and for any deterministic algorithm A on domain X that runs in space s and time t, the algorithm $B(A)$ runs in space $s \cdot O_\varepsilon\left((\log n)^{O(1/\delta)}\right)$ and time $m \cdot O_{\varepsilon,\delta}\left(1 + \frac{t}{n^2}\right)$.*

Throughout this section, we use the notation

$$A(q, \hat{x}) \in \{0,1\}^s$$

to denote the state of algorithm A when starting with the state $q \in \{0,1\}^s$ and executing on $\hat{x}$ received in a stream. By definition, there is an explicit algorithm that computes $A(q, \hat{x})$ in s space. We also use the shorthand $A(\hat{x})$ to denote $A(\emptyset, \hat{x})$. Notice that $A(x) = A(\emptyset, x)$ is simply the output state of the stream.

6.1 Statement of Algorithm

Let $\varepsilon > 0$. Throughout the section let $r = \log^{1/\delta} n$ and define $\ell := r^2 \log^4 n$. We also assume for simplicity that $\log_r n$ is an integer. We will also assume that the parameter $n > \max\{\exp(\exp(8\delta/\varepsilon)), \exp(1/\varepsilon)\}$ is sufficiently large, so that also $\ell < n$.

We begin by specifying the encoding $\mathsf{enc}(x)$ that Alice sends in the stream.

▶ **Definition 11** ($\mathsf{enc}(x)$)**.** *Let* $\mathsf{LDC}(x)$ *be a locally decodable code with* $N(n) = |\mathsf{LDC}(x)| = O_\varepsilon(n^{1+\delta})$ *and query complexity* $Q = O_\varepsilon(\log^{2/\delta} n)$ *satisfying the guarantees of Theorem 7 for* $\varepsilon/8$*. Then Alice sends* $\mathsf{enc}(x) := \mathsf{LDC}(x)^{\log^2 n \cdot M_n}$ *where* $M_n = (r \cdot \ell)^{\log_r n}$ *(that is,* $\mathsf{enc}(n)$ *is* $\log^2 n \cdot M_n$ *copies of* $\mathsf{LDC}(x)$*). In particular,* $m = |\mathsf{enc}(x)| = (r \cdot \ell)^{\log_r n} \cdot \log^2 n \cdot N(n) = O_\varepsilon(n^{4+6\delta})$.

Next, we describe Bob's streaming algorithm $B_A := B(A)$. Before stating it formally, we provide a high-level description.

Description

Let z be the incoming stream that Bob receives. At a high level, the algorithm B_A runs by generating many guesses for $A(x)$ along with confidences indicating how likely that guess was correct (depending on how much error is witnessed while generating the guess). At the end, the guess that has the most cumulative confidence is outputted.

For i, j where $j - i$ is an integer power of r, we define the algorithm $\mathsf{estA}(i, j, q)$ that has the following syntax:

- $\mathsf{estA}(i, j, q)$ takes as input two indices $i, j \in [n]$ along with a state $q \in \{0,1\}^s$ which represents a guess for the state of $A(q, x(i : j])$.
- It reads $N_{j-i} := (r \cdot \ell)^{\log_r(j-i)} \cdot N(n)$ many bits of z after which it outputs a guess for $A(q, x(i : j])$ along with a confidence.

In particular, $\mathsf{estA}(0, n, q)$ outputs a guess for $A(x)$ along with a confidence. By running $\mathsf{estA}(0, n, q)$ many times and aggregating the guesses weighted by the confidences, we obtain a final output.

At a high level, to compute $\mathsf{estA}(i, j, q)$, we break down the interval $(i : j]$ into r subintervals $(i_0 : i_1], (i_1 : i_2], \ldots, (i_{r-1} : i_r]$ where $i_a = i + a \cdot \frac{j-i}{r}$. We then recursively formulate guesses for each $A(x(1 : i_a])$ over many iterations by calling $\mathsf{estA}(i_{a-1}, i_a, q_{a-1})$ for some state q_{a-1}. The choice of a in each iteration must be randomized, so that the adversary cannot attack any single choice of a. More specifically, we will split up the N_{j-i} length input stream into ℓ chunks, each of length N_{j-i}/ℓ bits. Each chunk is split into r sections, each of size $N_{j-i}/(\ell \cdot r) = N_{(j-i)/r}$. In each chunk, we pick a random permutation $(\pi_1, \pi_2, \ldots, \pi_r)$ of $[r]$. In the a'th section of the chunk, we will compute on the subproblem corresponding to interval $(i_{\pi_a - 1} : i_{\pi_a}]$ by calling $\mathsf{estA}(i_{\pi_a - 1}, i_{\pi_a}, q_{\pi_a - 1})$.

Note that as the algorithm A is computed sequentially, computing a guess for $A(x[1 : i_a])$ requires having a starting state q_{a-1} for $A(x[1 : i_{a-1}])$. Thus, throughout the entire computation, we will keep track of the best guess and associated confidence for each of the r states $A(x[1 : i_a])$, denoted (q_a, c_a) and all initialized to $(\emptyset, 0)$.

These pairs (q_a, c_a) are maintained as follows. Whenever a recursive call to $\mathsf{estA}(i_{a-1}, i_a, q_{a-1})$ is completed, outputting $(\hat{q}, \hat{c})$, then: if $q_a = \hat{q}$ then the confidence c_a is increased by $\hat{c}$, and if $q_a \neq \hat{q}$ then the confidence c_a is decreased by $\hat{c}$. If the confidence c_a becomes negative, then we are more sure that the correct state is $\hat{q}$ rather than q_a: then q_a is replaced by $\hat{q}$ and the confidence is negated (so that it's positive). In this last case where q_a is replaced, we have no more reason to believe that further computations of $q_{a'}$, $a' > a$, are correct since they all depended on q_a, so we erase all such pairs $(q_{a'}, c_{a'})$ and reset them to $(\emptyset, 0)$. A key point in our analysis is to understand why this does not cause the error probability to accumulate.

Algorithm 2 Bob's Noise Resilient Algorithm B_A

input $n \in \mathbb{N}$ and stream $z \in \{0,1\}^m$.
function $\mathsf{estA}(i, j, q)$ ▷compute the state of A starting at state q between steps i and j
  if $j = i + 1$ **then**
    Read the next $|\mathsf{LDC}(x)|$ bits of the stream y.
    Using Theorem 7 compute guess $\hat{b}$ for $x[j]$ and confidence $\hat{c}$ in $s \cdot O_\varepsilon\left((\log n)^{O(1/\delta)}\right)$ bits of space.
    return $A(q, \hat{b}), c$.
  else
    Let $i_0 = i, i_1 = i + \frac{j-i}{r}, i_2 = i + \frac{2(j-i)}{r}, \ldots, i_r = j$.
    Initialize a list of pairs $(q_1, c_1), \ldots (q_r, c_r)$ each to $(\emptyset, 0)$. ▷cumulative best guesses and confidences
    for ℓ iterations **do**
      Let $\pi_1 \ldots \pi_r$ be a random permutation of $[r]$.
      Set $(q'_1, c'_1) = (q_1, c_1), \ldots (q'_r, c'_r) = (q_r, c_r)$.
      for $a \in [r]$ **do**
        Compute $(\hat{q}, \hat{c}) \leftarrow \mathsf{estA}(i_{\pi_a - 1}, i_{\pi_a}, q'_{\pi_a - 1})$ where $q'_0 := q$.
        if $\hat{q} = q'_{\pi_a}$ **then** ▷update the confidence on $\mathsf{estA}(i_{\pi_a}, i_{\pi_j + 1}, q_{\pi_a - 1})$
          $c_{\pi_a} \leftarrow c'_{\pi_a} + \hat{c}$
        else
          $c_{\pi_a} \leftarrow c'_{\pi_a} - \hat{c}$
          if $c_{\pi_a} < 0$ **then** ▷if the guess changes, reset its confidence
            $q_{\pi_a} \leftarrow \hat{q}$ and $c_{\pi_a} \leftarrow -c_{\pi_a}$.
      if some q_a was changed from its value at the beginning of the chunk **then**
        For all $i > a$, set $(q_i, c_i) \leftarrow (\emptyset, 0)$. ▷reset state and confidence for states with $i > a$
    return $q_r, \min(c_1 \ldots c_r)/\ell$

Initialize a pair (q, c) to $(\emptyset, 0)$.
for $\log^2 n$ iterations **do** ▷amplification step
  Let $(\hat{q}, \hat{c}) \leftarrow \mathsf{estA}(0, n, \emptyset)$
  if $\hat{q} = q$ **then**
    $c \leftarrow c + \hat{c}$
  else
    $c \leftarrow c - \hat{c}$
    if $c < 0$ **then**
      $q \leftarrow \hat{q}$ and $c \leftarrow -c$.
output q. ▷output $A(x)$

6.2 Proof of Theorem 1

The proofs that the algorithm satisfies the required space, computational and communication guarantees are essentially the same as in Section 5, but we reproduce them for completeness. The proof of correctness is somewhat more complicated but also follows the same general outline.

6.2.1 Algorithmic Complexity

The proofs of communication, time, and space complexities are omitted for the conference version.

6.2.2 Correctness

Next, we show correctness. Formally, correctness is shown by the following lemma.

▶ **Lemma 12.** *When at most* $\frac{1}{4} - \varepsilon$ *fraction of the stream* $\mathsf{enc}(x)$ *is corrupted, Algorithm 2 outputs* $A(x)$ *with probability* $\geq 1 - \exp\left(-\varepsilon^2 \log^2 n/32\right)$.

We prove the following statement which will easily imply Lemma 12. For a given i, j, q and associated string that is read by the algorithm in the computation, let the random variable $(q(i,j,q), c(i,j,q))$ denote the state and confidence after the computation $\mathsf{estA}(i,j,q)$. We define the *signed confidence*, denoted $c^{\pm}(i,j,q)$, to be defined as $+c(i,j,q)$ if $q(i,j,q) = A(q, x(i:j])$ and $-c(i,j,q)$ otherwise. For intuition, the more positive $c^{\pm}(i,j,q)$ is, the more correct with higher confidence the output $(q(i,j,q), c(i,j,q))$ is, whereas the more negative $c_{\pm}(i,j,q)$ is, the more $q(i,j,q)$ is incorrect with high confidence. So, $c^{\pm}(i,j,q)$ gives us a scaled measure of how correct the output of $\mathsf{estA}(i,j,q)$ is.

▶ **Lemma 13.** *For any* $0 \leq i < j \leq n$, $q \in \{0,1\}^s$, *and* $e \in \mathbb{R}$, *given* $\leq \frac{1}{4} - \frac{\varepsilon}{4} - \frac{\varepsilon}{4} \cdot \frac{\log(j-i)}{\log n} - e$ *fraction of corruptions in the bits read by the computation of* $\mathsf{estA}(i,j,q)$, *we have* $\mathbb{E}[c^{\pm}(i,j,q)] > e$.

Proof. The proof of this lemma is omitted for the conference version. ◀

Proof of Lemma 12. In the special case where $i = 0, j = n, q = \emptyset$ and there are $\leq \frac{1}{4} - \frac{\varepsilon}{2} - e < \frac{1}{4} - \frac{\varepsilon}{4} - \frac{\varepsilon}{4} \cdot \frac{\log(j-i)}{r} - e$ errors (where the inequality follows because $\log n < \varepsilon \cdot r$), by Lemma 13 we have that $\mathbb{E}[c^{\pm}(0,n,\emptyset)] > e$.

Then, the final amplification step of the protocol computes the pair $(q(0,n,\emptyset), c(0,n,\emptyset))$ for $\log^2 n$ times, where in each chunk i we denote the fraction of error to be $\left(\frac{1}{4} - \frac{\varepsilon}{2} - e_i\right)$, where $\sum_i e_i \geq \frac{\varepsilon \log^2 n}{2}$ since we assumed the total error was $\leq \frac{1}{4} - \varepsilon$. Also, the protocol outputs $A(x)$ if and only if $\sum_i c_i^{\pm}(0,n,\emptyset)$ (denoting the value of $c^{\pm}(0,n,\emptyset)$ in the i'th chunk) is positive. By Azuma's inequality,

$$Pr\left[\sum_i c_i^{\pm}(0,n,\emptyset) > 0\right] \geq Pr\left[\sum_i \left(c_i^{\pm}(0,n,\emptyset) - e_i\right) > -\frac{\varepsilon \log^2 n}{2}\right] \geq 1 - \exp\left(-\frac{\varepsilon^2 \log^2 n}{32}\right).$$

◀

This concludes the proof of Theorem 1.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. *Analyzing Graph Structure via Linear Measurements*, pages 459–467. SIAM, 2012. doi:10.1137/1.9781611973099.40.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, 1996. doi:10.1145/237814.237823.

3 Omri Ben-Eliezer, Rajesh Jayaram, David P Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. *ACM Journal of the ACM (JACM)*, 69(2):1–33, 2022. doi:10.1145/3498334.

4 Zvika Brakerski and Yael Tauman Kalai. Efficient Interactive Coding against Adversarial Noise. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 160–166, 2012. doi:10.1109/FOCS.2012.22.

5 Zvika Brakerski and Moni Naor. Fast Algorithms for Interactive Coding. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 443–456, USA, 2013. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973105.32.

6 Mark Braverman. Towards Deterministic Tree Code Constructions. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 161–167, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2090236.2090250.

7 Mark Braverman and Klim Efremenko. List and Unique Coding for Interactive Communication in the Presence of Adversarial Noise. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 236–245, Los Alamitos, CA, USA, October 2014. IEEE Computer Society. doi:10.1109/FOCS.2014.33.

8 Mark Braverman and Anup Rao. Towards Coding for Maximum Errors in Interactive Communication. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 159–166, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1993636.1993659.

9 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002. doi:10.1007/3-540-45465-9_59.

10 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004. Automata, Languages and Programming. doi:10.1016/S0304-3975(03)00400-6.

11 Jiecao Chen and Qin Zhang. Bias-aware sketches. *arXiv preprint*, 2016. arXiv:1610.07718.

12 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. doi:10.1016/j.jalgor.2003.12.001.

13 Varsha Dani, Thomas P. Hayes, Mahnush Movahedi, Jared Saia, and Maxwell Young. Interactive Communication with Unknown Noise Rate, 2015. arXiv:1504.06316.

14 Irit Dinur, Shai Evra, Ron Livne, Alexander Lubotzky, and Shahar Mozes. Locally testable codes with constant rate, distance, and locality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 357–374, 2022. doi:10.1145/3519935.3520024.

15 Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM Journal on Computing*, 40(4):1154–1178, 2011. doi:10.1137/100804322.

16 Klim Efremenko. 3-query locally decodable codes of subexponential length. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 39–44, 2009. doi:10.1145/1536414.1536422.

17 Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Maximal Noise in Interactive Communication Over Erasure Channels and Channels With Feedback. *IEEE Trans. Inf. Theory*, 62(8):4575–4588, 2016. doi:10.1109/TIT.2016.2582176.

18 Klim Efremenko, Bernhard Haeupler, Yael Tauman Kalai, Gillat Kol, Nicolas Resch, and Raghuvansh R Saxena. Interactive coding with small memory. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3587–3613. SIAM, 2023. doi:10.1137/1.9781611977554.CH137.

19 Klim Efremenko, Gillat Kol, and Raghuvansh R. Saxena. Binary Interactive Error Resilience Beyond $1/8$ (or why $(1/2)^3 > 1/8$). In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 470–481, 2020. doi:10.1109/FOCS46700.2020.00051.

20 Philippe Flajolet. Approximate counting: a detailed analysis. *BIT Numerical Mathematics*, 25(1):113–134, 1985. doi:10.1007/BF01934993.

21 G. Forney. Generalized minimum distance decoding. *IEEE Transactions on Information Theory*, 12(2):125–131, 1966. doi:10.1109/TIT.1966.1053873.

22 Sumegha Garg, Pravesh K Kothari, Pengda Liu, and Ran Raz. Memory-sample lower bounds for learning parity with noise. *arXiv preprint*, 2021. arXiv:2107.02320.

23 Ran Gelles. Coding for Interactive Communication: A Survey. *Foundations and Trends® in Theoretical Computer Science*, 13:1–161, January 2017. doi:10.1561/0400000079.

24 Ran Gelles and Bernhard Haeupler. Capacity of Interactive Communication over Erasure Channels and Channels with Feedback. *SIAM Journal on Computing*, 46:1449–1472, January 2017. doi:10.1137/15M1052202.

25 Ran Gelles, Bernhard Haeupler, Gillat Kol, Noga Ron-Zewi, and Avi Wigderson. *Towards Optimal Deterministic Coding for Interactive Communication*, pages 1922–1936. SIAM, 2016. doi:10.1137/1.9781611974331.ch135.

26 Ran Gelles and Siddharth Iyer. Interactive coding resilient to an unknown number of erasures. *arXiv preprint*, 2018. arXiv:1811.02527.

27 Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Information Processing Letters*, 43(4):169–174, 1992. doi:10.1016/0020-0190(92)90195-2.

28 Mohsen Ghaffari and Bernhard Haeupler. Optimal Error Rates for Interactive Coding II: Efficiency and List Decoding. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, December 2013. doi:10.1109/FOCS.2014.49.

29 Meghal Gupta, Venkatesan Guruswami, and Mihir Singhal. Tight bounds for stream decodable error-correcting codes, 2024. doi:10.48550/arXiv.2407.06446.

30 Meghal Gupta and Rachel Yun Zhang. Efficient interactive coding achieving optimal error resilience over the binary channel. *arXiv preprint*, 2022. doi:10.48550/arXiv.2207.01144.

31 Meghal Gupta and Rachel Yun Zhang. The Optimal Error Resilience of Interactive Communication Over Binary Channels. In *Symposium on Theory of Computing, STOC 2012, New York, NY, USA, June 20 - June 24, 2022*, STOC '22. ACM, 2022. doi:10.1145/3519935.3519985.

32 Venkat Guruswami. Error-correcting codes: Constructions and algorithms, lecture no. 11, 2006. URL: http://www.cs.washington.edu/education/courses/533/06au/.

33 Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 181–190, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335305.335327.

34 Bernhard Haeupler. Interactive Channel Capacity Revisited. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 226–235, 2014. doi:10.1109/FOCS.2014.32.

35 Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.

36 Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208, 2005. doi:10.1145/1060590.1060621.

37 William Johnson and Joram Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, January 1984. doi:10.1090/conm/026/737400.

38 Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. *J. ACM*, 61(5), September 2014. doi:10.1145/2629416.

39 Chaoyi Ma, Haibo Wang, Olufemi Odegbile, and Shigang Chen. Noise measurement and removal for data streaming algorithms with network applications. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2021. doi:10.23919/IFIPNETWORKING52078.2021.9472845.

40 Andrew McGregor, Atri Rudra, and Steve Uurtamo. Polynomial fitting of data streams with applications to codeword testing. *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, 9, March 2011. doi:10.4230/LIPIcs.STACS.2011.428.

41 Morteza Monemizadeh and David P Woodruff. 1-pass relative-error lp-sampling with applications. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1143–1160. SIAM, 2010. doi:10.1137/1.9781611973075.92.

42 Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978. doi:10.1145/359619.359627.

43 David E Muller. Application of boolean algebra to switching circuit design and to error detection. *Transactions of the IRE professional group on electronic computers*, 3:6–12, 1954. doi:10.1109/IREPGELC.1954.6499441.

44 N. Nisan. Pseudorandom generators for space-bounded computations. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 204–212, New York, NY, USA, 1990. Association for Computing Machinery. doi:10.1145/100216.100242.

45 I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. doi:10.1137/0108018.

46 Irving S Reed. A class of multiple-error-correcting codes and the decoding scheme. *IEEE Transactions on Information Theory*, 4(4):38–49, 1954. doi:10.1109/TIT.1954.1057465.

47 Atri Rudra and Steve Uurtamo. Data stream algorithms for codeword testing. In *International Colloquium on Automata, Languages, and Programming*, pages 629–640. Springer, 2010. doi:10.1007/978-3-642-14165-2_53.

48 Leonard J. Schulman. Communication on noisy channels: a coding theorem for computation. In *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, pages 724–733, 1992. doi:10.1109/SFCS.1992.267778.

49 Leonard J. Schulman. Deterministic Coding for Interactive Communication. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 747–756, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167279.

50 Leonard J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. doi:10.1109/18.556671.

51 Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi:10.1002/j.1538-7305.1948.tb01338.x.

52 Sergey Yekhanin et al. Locally decodable codes. *Foundations and Trends® in Theoretical Computer Science*, 6(3):139–255, 2012. doi:10.1561/0400000030.