Tight Bounds for Stream Decodable Error-Correcting Codes

Meghal Gupta
UC Berkeley, CA, USA

Venkatesan Guruswami
UC Berkeley, CA, USA

Mihir Singhal
UC Berkeley, CA, USA

Abstract

In order to communicate a message over a noisy channel, a sender (Alice) uses an error-correcting code to encode her message, a bitstring x, into a codeword. The receiver (Bob) decodes x correctly whenever there is at most a small constant fraction of adversarial errors in the transmitted codeword. We investigate the setting where Bob is restricted to be a low-space streaming algorithm. Specifically, Bob receives the message as a *stream* and must process it and write x in order to a write-only tape while using low (say polylogarithmic) space. Note that such a primitive then allows the execution of any downstream streaming computation on x.

We show three basic results about this setting, which are informally as follows:

(i) There is a stream decodable code of near-quadratic length, resilient to error-fractions approaching the optimal bound of 1/4.

(ii) There is no stream decodable code of sub-quadratic length, even to correct any small constant fraction of errors.

(iii) If Bob need only compute a private linear function of the bits of x, instead of writing them all to the output tape, there is a stream decodable code of near-linear length.

Our constructions use locally decodable codes with additional functionality in the decoding, and (for the result on linear functions) repeated tensoring. Our lower bound, which rather surprisingly demonstrates a strong information-theoretic limitation originating from a computational restriction, proceeds via careful control of the message indices that may be output during successive blocks of the stream, a task complicated by the arbitrary state of the decoder during the algorithm.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Coding theory, Streaming computation, Locally decodable code, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.CCC.2025.13

Related Version *Full Version*: https://arxiv.org/abs/2407.06446

Funding Research supported in part by NSF GRFP Fellowships (for M.G and M.S), a Simons Investigator award (for V.G), and NSF grans CCF-2210823 and CCF-2211972.

1 Introduction

Consider the following task: a sender wishes to communicate a message to a receiver that it receives and processes bit-by-bit. This scenario arises, for instance, in automatic control applications, where a device receives an incoming stream of instructions that it executes in sequence. Concretely, consider a small satellite receiving instructions from a large control center on the ground. The control center wants to send the satellite instructions in a way that satisfies two properties:

- **Error resilience.** The satellite should execute the correct set of instructions even if a constant fraction of the transmission is corrupted.
- **Low-memory decoding.** The satellite should be able to process the instructions in order while only using limited space (significantly less than the total length of the instructions).

Sending the list of instructions $x_1 \dots x_n$ directly, although easy to process and execute one-by-one, offers no resilience against errors. On the other hand, encoding $x_1 \dots x_n$ into $\mathsf{ECC}(x_1 \dots x_n)$ with a standard error-correcting code [29, 17] is resilient to error, but requires the receiver to store the whole stream to decode, and thus use too much space. An intermediate approach would be to encode the individual instructions each by error-correcting codes as $\mathsf{ECC}(x_1)\mathsf{ECC}(x_2) \dots \mathsf{ECC}(x_n)$. However, this does not withstand a constant overall fraction of corruption: the adversary can corrupt $\mathsf{ECC}(x_1)$ entirely, using only a $1/n$ fraction of corruption, and thus never allow the satellite to recover x_1. We remark that is reminiscent of the issue faced by the naive approach, that encodes each round separately, in protecting interactive communication against errors [26]. What we would like is a code that encodes the message globally but can be decoded in low space as the encoded message arrives.

Stream-decodable codes. This motivates the notion of a *stream-decodable* error-correcting code. In this model, we require that the receiver can output the entire message $x_1 \dots x_n$ when any small fraction ρ of the message is adversarially corrupted while using low space (for example, polylog(n) space) to process the transmission bit-by-bit. More formally, a stream decodable code has the following two components:

- An encoding function $\mathsf{enc} : \{0,1\}^n \to \{0,1\}^{m(n)}$.
- A randomized decoding algorithm $\mathsf{dec} : \{0,1\}^{m(n)} \to \{0,1\}^n$ that uses $s(n)$ space ($s(n)$ is much smaller than n: for instance, $s(n) = \text{polylog}(n)$). For all x, and for any $z \in \{0,1\}^{m(n)}$ within Hamming distance $\rho \cdot m(n)$ of $\mathsf{enc}(x)$, it should hold that $\mathsf{dec}(z)$ outputs x with high probability.

It is not clear that such codes should exist at all for any $s(n) = o(n)$, even for small error rates ρ and an arbitrary communication blow-up $m(n)$. In particular, a standard error-correcting code could require storing the full encoding at once to process, and so it requires $s(n) = n$.

Our first result constructs stream-decodable codes that achieve the following parameters (see Theorem 1 for a precise statement).

- Error resilience of $\rho = \frac{1}{4} - \varepsilon$ for any $\varepsilon > 0$, matching the best possible error resilience of standard error-correcting codes.
- Near-quadratic blow-up in communication: $m(n) = \frac{n^{2+r(s(n))}}{s(n)}$ (here $r(s(n))$ is small – typically $o(1)$, but when $s(n) = \log(n)^t$, then $r(s(n)) = 1/t$). This is a larger blow-up in communication than incurred by standard error-correcting codes.

The construction itself is quite simple: it encodes the message by a locally decodable code of near-linear length, and repeats the encoding $O(n)$ times. The more interesting part is the corresponding decoding algorithm for Bob. To this end, we work with stronger local decoding guarantees, specifically having access to soft information for unique decoding, and local list decoding with advice from close to $1/2$ errors.

A matching lower bound. Our next result demonstrates, surprisingly, that the communication blowup of our codes is essentially optimal: any stream-decodable code requires transmission length $m(n) = \Omega\left(\frac{n^2}{s(n)}\right)$, in contrast to the standard error-correction model. (See Theorem 2 for the precise statetemt.) This result is surprising and notable because

it obtains a strong lower bound on an information-theoretic quantity (the codeword blow-up) leveraging the computational restriction of space-boundedness. The lower bound is established by carefully controlling the set of message indices that the decoder can output when processing successive blocks of sub-linear size of the stream. It is also interesting to contrast this with the earlier mentioned setting of interactive communication, which despite the challenge of encoding the global conversation that is only available in local fragments, admits non-trivial schemes with a constant factor communication blow-up, from Schulman's seminal work [26] and many follow-ups (surveyed in [10]).

Comparison to [12]. Our work can be viewed as a strengthening of and resolution to most of the open problems in [12]. The problem is framed slightly differently in their work: in [12], instead of outputting $x_1 \dots x_n$, the decoder is only required to output a single bit $f(x_1 \dots x_n)$. Here, the function f represents the output of an arbitrary streaming algorithm performed on $x_1 \dots x_n$, so it must be possible to compute in $s(n)$ space in a streaming manner. The function f is unknown to Alice (or else she could simply send the value $f(x_1 \dots x_n)$ to Bob) but known to the adversary causing the errors. One could imagine, for example, that f is an arbitrary linear function of $x_1 \dots x_n$, or one's physical location after executing some (possibly non-commutative) sequence of movements $x_1 \dots x_n$.

For this problem, [12] provide a scheme requiring slightly larger than $O(n^4)$ encoding length. Specifically, if $s(n) = \log(n)^t$, their code requires $n^{4+O(1/t)}$ space. Our notion of a stream-decodable code is necessarily stronger: that is, if the decoder can write $x_1 \dots x_n$ to an output tape in that order, they can also compute the output of any streaming algorithm f in low space. In this sense, we provide a *generic* "front-end" error-correction scheme that enables executing any streaming task in a noise-resilient manner. Further, our construction improves upon the encoding length of the one in [12], providing a nearly quadratic-length code for their problem.

Gupta and Zhang [12] also specifically investigate the scenario where Alice knows beforehand that Bob's function f is a linear function of $x_1, \dots, x_n$. For this restricted class of functions, they demonstrate a scheme that uses slightly larger than $O(n^2)$ encoding length.

Near-linear length code for stream computation of linear functions. Our final result, stated precisely as Theorem 3, is a new scheme for stream-decoding linear functions. Specifically, we improve upon Gupta and Zhang's result, demonstrating a scheme that requires only near-linear communication in n for computing linear functions. This is achieved using a tensor product of locally decodable codes for the encoding, and a careful recursive decoding approach to recover the desired linear function of the message.

1.1 The model definition

Before we provide the technical statements of our three main results, let us formally define the model of stream decodable codes. A $(\rho, m(n), s(n))$-*stream coding scheme* with probability p of success consists of the following:

- An explicit family of encoding algorithms $\mathsf{enc} = \{\mathsf{enc}_n : \{0,1\}^n \to \{0,1\}^{m(n)}\}$ with encoding time poly(n).
- An explicit family of randomized decoding algorithms $\mathsf{dec} = \{\mathsf{dec}_n : \{0,1\}^{m(n)} \to \{0,1\}^n\}$ that read the input in a stream and are permitted $s(n)$ memory and poly(n) time. The output is written to a one-way (left-to-right) write-only tape that writes only left-to-right. Whenever the Hamming distance $\Delta(z, \mathsf{enc}(x)) < \rho m$, $\mathsf{dec}(z)$ correctly outputs x with probability p.

More generally, a $(\rho, m(n), s(n))$-stream coding scheme for a family of functions $\mathcal{F} = \{f : \{0,1\}^* \to \{0,1\}^*\}$ consists of the following similar components, with the same time and space guarantees as above:

- An explicit family of encoding algorithms $\mathsf{enc}^{(\mathcal{F})} = \{\mathsf{enc}_n^{(\mathcal{F})} : \{0,1\}^n \to \{0,1\}^{m(n)}\}$.
- For each $f \in \mathcal{F}$, an explicit family of randomized decoding algorithms $\mathsf{dec}^{(f)} = \{\mathsf{dec}_n^{(f)} : \{0,1\}^{m(n)} \to \{0,1\}^n\}$ that read the input in a stream and outputs to one-way write-only tape. Whenever the Hamming distance $\Delta(z, \mathsf{enc}(x)) < \rho m$, the decoder $\mathsf{dec}^{(f)}(z)$ outputs $f(x)$ with probability p.

We emphasize that the encoding has no knowledge of f, only of the family $\mathcal{F}$, while the decoding algorithm must succeed for all f.

1.2 Our results

In this section, we formally state our results. In this section, when we use the phrase "absolute constant" to describe any parameter, we mean that any asymptotic notation henceforth may have constants depending on that absolute constant.

The first result is a stream decodable error-correcting code incurring approximately quadratic blow-up in communication.

▶ **Theorem 1.** *Fix any absolute constant $\varepsilon > 0$. Then, for some large enough $C = C(\varepsilon)$ and $c = c(\varepsilon)$, the following hold.*

- *If $s(n) = (\log n)^t$ for some absolute constant $t > C$, then there is a $\left(\frac{1}{4} - \varepsilon, n^{2+c/t}, s(n)\right)$-stream coding scheme.*
- *For any function $s(n) = (\log n)^{\omega(1)}$, there is a $\left(\frac{1}{4} - \varepsilon, \frac{n^{2+o(1)}}{s(n)}, s(n)\right)$-stream coding scheme. (Here, the implicit constants in the $o(1)$ may depend on those in the $\omega(1)$.)*

Both schemes succeed with probability $1 - \frac{1}{n^{\omega(1)}}$.

The second result establishes that Theorem 1 is essentially optimal. That is, any encoding of a message that permits a low-space streaming algorithm to decode requires $\Omega\left(\frac{n^2}{s(n)}\right)$ length. We view this as one of the major surprises and contributions of this work.

▶ **Theorem 2.** *Fix an absolute constant $\rho > 0$ and let the space for the decoding algorithm be $s(n) \geq \log n$. Suppose there is a $(\rho, m, s(n))$-coding scheme for streams that succeeds with probability at least $1 - \frac{1}{2n^2}$. Then, $m = \Omega\left(\frac{n^2}{s(n)}\right)$.*

The final result states that the encoding length can be made almost linear for stream coding schemes that compute a linear function $f(x)$. Here, the decoder need only output a private linear function f of the input bits rather than the entire input. When $s(n) = n^\delta$ for sufficiently small n, this can even be made exactly linear, which is optimal.

▶ **Theorem 3.** *Fix any absolute constant $\varepsilon > 0$. Then, for some large enough $C = C(\varepsilon)$ and $c = c(\varepsilon)$, the following hold.*

- *If $s(n) = (\log n)^t$ for some absolute constant $t > C$, then there is a $\left(\frac{1}{4} - \varepsilon, n^{1+c/\sqrt{t}}, s(n)\right)$-stream coding scheme for the family of linear functions.*
- *If $s(n) = \Omega(n^\delta)$ for some absolute constant $\delta > 0$, then there is a $\left(\frac{1}{4} - \varepsilon, O(n), s(n)\right)$-coding scheme for the family of linear functions.*

Both schemes succeed with probability $1 - \frac{1}{n^{\omega(1)}}$.

1.3 Discussion and further directions

Tightening lower order terms

Both Theorem 1 and Theorem 3 construct codes that are not optimal in lower-order terms. It may be possible to construct stream coding schemes of exactly length $O\left(\frac{n^2}{s(n)}\right)$ and stream coding schemes for linear functions of length $O(n)$. This is an interesting direction for future work.

Specifically, in the case of linear functions, it may be possible to construct constant rate codes. Interestingly, we can pose this question for an even more restrictive class of functions than linear functions: the class of index functions. These are the functions $f_i(x) = x_i$ for all i. We do not even know if constant rate stream coding schemes exist here, when $s(n)$ is sufficiently small, say polylog(n).

One simple strategy is to encode with a locally decodable code requiring $Q = s(n)^{O(1)}$ queries to recover an index. The decoder can then ignore all the indices except the Q it needs to recover the target index, and in poly$(Q) = s(n)$ space, recover any individual index of the message. Indeed, for our constructions in both Theorem 1 and Theorem 3, this procedure is an important primitive. Unfortunately, the best locally decodable codes for polylogarithmic locality have super-linear encoding length [32], and so are not constant rate.

However, this is not necessarily the only way. Barring a constant rate construction of polylog(n)-query locally decodable codes, can we construct constant rate stream decoding schemes? We remark that if one removes the streaming requirement and only requires that the decoder be low space with arbitrary queries, constant rate codes are known [30, 13].

Improvements to the lower bound

We will discuss a few potential strengthenings to Theorem 2.

The work of [12] initially proposes the model of stream coding schemes where the decoder need only output $f(x_1 \dots x_n)$, for an arbitrary choice of Boolean function f that can be computed by receiving $x_1 \dots x_n$ in a stream in $s(n)$ space. The simplest way to accomplish this task is to compute $x_1 \dots x_n$ in order and perform the streaming computation of f as each bit is discovered. Our lower bound shows that this method requires encoding length $\Omega\left(\frac{n^2}{s(n)}\right)$, but there could be a different way. Nonetheless, we conjecture that there is a Boolean function computable by a $s(n)$-space streaming algorithms for which the lower bound of $\Omega\left(\frac{n^2}{s(n)}\right)$ encoding length holds. Candidates for such a function might be some form of group product over non-abelian groups.

Secondly, our lower bound in Theorem 2 only disproves stream coding schemes where the decoder outputs x with probability $1 - \frac{1}{\text{poly}(n)}$. However, random guessing only outputs x correctly with probability $\frac{1}{2^n}$. We conjecture that a stream coding scheme requires $\Omega\left(\frac{n^2}{s(n)}\right)$ space to output x even if we only require the success probability to be $\frac{1}{\text{poly}(n)}$.

1.4 Related Work

Aside from the connection to [12], we discuss the relation of our work different models of error-correcting codes and to streaming algorithms.

Efficiency of error-correction algorithms. Our work explores the model of stream decodable error-correcting codes where the decoder must be low-space and read the codeword bits in a single pass. Without the latter restriction, it is known how to construct asymptotically good

codes for which a *logspace* decoder can correct a constant fraction of errors, and in fact one can also have a logspace encoder [30, 13]. Note that the decoder is not restricted to a single pass on the codeword. Since one typically receives a communicated codeword as a stream, a natural question is whether such results extends to low space decoding in a stream. This is our focus, and we show that for error-correction in this setting, one cannot have codes of constant rate, and in fact a near quadratic blow-up in message length becomes necessary.

Codes against streaming channels. Streaming in the context of error-correction has previously been considered for *channels* which are low-space (say logarithmic in the code length) and cause errors in an online fashion as they read the input codeword in a single pass. This was part of a more general line of work on coding against computationally bounded channels whose systematic study was initiated in [15]. List-decodable codes with optimal rate against such channels were constructed in [27] for all error fractions $p \in [0, 1/2)$, and their decoding time improved in [20]. More recently and surprisingly, even unique-decodable codes with optimal rate (for at most fractions $p < 1/4$ of errors caused by a streaming channel) were constructed in [28].

There is also beautiful work on causal channels, which must read the codeword in one pass, but there is no space restriction [5]. In contrast, our work is in the model where the *receiver*, as opposed to the channel, is the computationally restricted party.

Additionally, the authors of [8] consider a related version of the problem we consider, where the encoder also receives the message as a stream rather than all at once, but the encoder and decoder are permitted shared randomness. In this setting, they show that it is possible to achieve a constant-rate encoding with any constant fraction of errors less than 1 (over large alphabets).

Locally decodable codes. One specific type of error-correcting codes related to our result is that of *locally decodable codes*. Locally decodable codes [33] 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. In constrast, for us, the decoder must be able to piece the entire stream with the relaxation that the decoder accesses the entire encoding via a stream rather than via query access. Locally decodable codes have been constructed in a variety of different parameter regimes, including constant query [7, 6] and rates approaching 1 [19]. In our work, we will use Reed-Muller codes [24, 25] that achieve polylog(n) query complexity and slightly super-linear block length.

As discussed in Section 1.3, our work also connects to locally decodable codes as a relaxation of the query model. Specifically, q-query locally decodable codes are poly(q) space stream coding schemes for the family of linear functions (as long as the locally decodable code permits poly(q) decoding space). Thus, our model can be viewed as a simpler setting than local decoding in which to construct high rate codes. In particular, the existence of constant rate stream decodable codes for index functions may be easier to resolve than constant rate polylogarithmic-query locally decodable codes.

Streaming Algorithms. The algorithms in this work are *streaming algorithms* for processing noisy encoded communication.

Streaming algorithms are a prolific field of research with algorithms for a multitude of problems, including approximate counting [23] on approximate counting, heavy hitters [3], ℓ_p approximation [1, 22, 18], and identifying a nonzero entry in a vector (for turnstile

algorithms) [22]. Many works, for example [9, 4, 21, 2], also consider the problem of processing *noisy* data using streaming algorithms. [9] shows a memory lower bound for learning a parity function with noisy samples of random linear equations.

However, the typical streaming setting is quite different from our setting. The algorithms mentioned above are used to process adversarial or "random" data. Our algorithms on the other hand process carefully formatted streams in the presence of communication noise, rather than sample or data noise. Our streaming algorithms are for processing formatted communication rather than processing data.

2 Preliminaries

Notation

- The function log is in base 2 unless otherwise specified.
- The set $[n]$ denotes the integers $1 \dots n$.
- Given a tuple T and element i, the expression $T|i$ denotes i concatenated to T.
- The phrase "with high probability in n" means with probability at least $1 - \frac{1}{n^{\omega(1)}}$.
- We use $\Delta(x, y)$ to denote the Hamming distance between two strings $x, y \in (\Sigma \cup \perp)^n$, and $\delta(x, y)$ to denote the relative distance between them (i.e. $\delta(x, y) = \frac{1}{n} \cdot \Delta(x, y)$). Any element of Σ is considered distance $\frac{1}{2}$ from $\perp$.
- For clarity, we will often omit floor and ceiling signs where they would technically be necessary.

▶ **Lemma 4** (Tail bound for k-wise independent random variables). *Let $k > 4$ be an even integer. Suppose $X_1, X_2, \dots, X_n$ are k-wise independent random variables taking values in $[0, 1]$. Let $Z = \sum_i X_i$ and $\mu = \mathbb{E}[Z]$. Then*

$$\Pr[|Z - \mu| \geq A] \leq 8 \cdot \left(\frac{k\mu + k^2}{A^2}\right)^{k/2} .$$

2.1 Error-correcting codes

We begin with some results about error-correcting codes. We first state a theorem detailing the existence of distance $1 - 1/|\mathbb{K}| - \varepsilon$ codes that are efficiently encodable and efficiently decodable. It is standard, and based on GMD decoding of concatenated codes with an outer code approaching the Singleton bound (like Reed-Solomon or algebraic-geometric codes), and a small inner code of relative distance close to $(1 - 1/|\mathbb{K}|)$ (see for instance [14, Chap.14]).

▶ **Theorem 5.** *For every $\varepsilon > 0$ and every finite field $\mathbb{K}$, there exists an explicit systematic linear error-correcting code $\mathsf{ECC}_\varepsilon = \{\mathsf{ECC}_{\varepsilon,n} : \mathbb{K}^n \to \mathbb{K}^m\}_{n \in \mathbb{N}}$ with relative distance at least $1 - 1/|\mathbb{K}| - \varepsilon$ and $m \leq n/\varepsilon^{O(1)}$, and a $O_\varepsilon(n^2)$-time and space decoding algorithm $\mathsf{DEC}_\varepsilon : \mathbb{K}^m \to \mathbb{K}^n$, such that for any $x \in \mathbb{K}^n$ and $w \in \mathbb{K}^m$ satisfying $\delta(\mathsf{ECC}_\varepsilon(x), w) < (1 - 1/|\mathbb{K}|)(1 - \varepsilon)/2$, it holds that $x = \mathsf{DEC}_\varepsilon(w)$.*

Our constructions will use Reed-Muller codes (based on evaluations of multivariate polynomials) concatenated with the codes from Theorem 5. In order to locally decode these codes, we will correct them along lines for which we would need to run list decoding algorithms for concatenated codes with outer Reed-Solomon codes. The following list decoding result for such codes is standard, and based on list-decoding the inner codes by brute force and list-recovering the outer Reed-Solomon codes; see for example [16]. (Better parameters are possible, but having poly(ε) rate and poly($1/\varepsilon$) output list size suffices for our purposes.)

▶ **Theorem 6.** *Let $\varepsilon > 0$. Let C be a concatenated code with outer Reed-Solomon code over $\mathbb{F}_q$ of rate $(\frac{\varepsilon}{4})^4$ and an inner code of relative distance at least $\frac{1}{2} - \frac{\varepsilon^2}{16}$. Then C can be list-decoded in $\mathrm{poly}(q)$ time from a fraction $(1-\varepsilon)/2$ of errors with an output list size of $64/\varepsilon^3$.*

3 Locally decodable/correctable codes

In this section, we introduce the locally decodable/correctable codes that will form the backbone of the constructions in our paper. There are two theorems we require, one for Section 4 and one for Section 6. Each of our codes will require a feature besides just correctness of local decoding/correcting when the distance to a codeword is small. Both proofs are omitted for this conference version.

Our first code for binary alphabets has two additional requirements. It requires that the decoder output a probability of each output 0 and 1 rather than only one of them (smoothness). This requirement is similar to list decoding with confidences from [12], and we adapt their proof below. Secondly, it has a local decoding with advice guarantee. To establish this notion, we use ideas similar to [31] and the locally list-decodable codes of [11].

Our second code requires only a smoothness guarantee for local decoding. However, it is in the regime with large alphabet and non-constant ε, and the code is required to be linear.

▶ **Theorem 7** (Binary locally decodable code)**.** *Fix an arbitary $\varepsilon > 0$. Let $Q = Q(n) \in [(\log n)^{100}, n]$. There is a code $\mathsf{LDC} : \{0,1\}^n \to \{0,1\}^N$ that satisfies the following properties:*

- **Length:** *The length $N = N(n) \leq n \cdot (\log_Q n)^{100 \log_Q n}$.*
- **Distance:** *For any $x \neq y \in \{0,1\}^n$, it holds that $\delta(\mathsf{LDC}(x), \mathsf{LDC}(y)) \geq \frac{1}{2} - \varepsilon$.*
- **Smooth local decoding/correcting:** *There exists a randomized algorithm $\mathcal{A}$ that on input $i \in [n]$ (resp. input $i \in [N]$) non-adaptively queries Q bits of the encoding and runs in $O(Q^3)$ time and space and achieves the following guarantee. For any word $w \in \{0,1\}^N$, with high probability in n, the algorithm outputs probabilities $p(b)$ for $b \in \{0,1\}$ that satisfy $p(0) + p(1) = 1$ and $p(x_i) > 1 - 2\delta(w, \mathsf{LDC}(x)) - \varepsilon$ (resp. $p(\mathsf{LDC}(x)_i) > 1 - 2\delta(w, \mathsf{LDC}(x)) - \varepsilon$).*
- **Local decoding with advice:** *There exists a randomized algorithm that on input $i \in [n]$ queries Q bits of the encoding (non-adaptively) and runs in $\mathrm{poly}(Q)$ time and space, and a distribution $\mathcal{D}$ on $[N]^u$ (independent of i) for some $u = O(Q)$ (that is, subsets of indices of size u), which does the following. For any word $w \in \{0,1\}^N$ satisfying $\delta(w, \mathsf{LDC}(x)) < \frac{1}{2} - \varepsilon$, it outputs x_i with high probability in n when additionally given $\mathsf{LDC}(x)_{d_1} \dots \mathsf{LDC}(x)_{d_u}$ for $d_1 \dots d_s \sim \mathcal{D}$.*

The proof can be found in the full version of the paper on arXiv.

We next turn to the proof of the large alphabet version of Theorem 7. Here, we will only need the smooth local decoding guarantee.

▶ **Theorem 8** (Large alphabet locally decodable code)**.** *Let $\varepsilon > 0$ and let $\mathbb{K}$ be a field of the form $\mathbb{F}_{2^k}$ where $2^k > \varepsilon^{-10}$, and let $Q = Q(n) \in [(\varepsilon^{-1} \log n)^{100}, n]$. There is a linear code $\mathsf{LDC} : \mathbb{K}^n \to \mathbb{K}^N$ that satisfies the following properties:*

- **Length:** *The length $N = N(n)$ satsifies $N \leq n \cdot (\varepsilon^{-1} \log_Q(n))^{100 \log_Q(n)}$.*
- **Distance:** *For any $x \neq y \in \mathbb{K}^n$, it holds that $\delta(\mathsf{LDC}(x), \mathsf{LDC}(y)) \geq 1 - \varepsilon^6$.*
- **Large alphabet smooth local decoding/correcting:** *There exists a randomized algorithm $\mathcal{A}$ that on input $i \in [n]$ (resp. input $i \in [N]$) queries Q bits (non-adaptively) of the encoding and runs in $O(Q^3)$ time and space and does the following. For any word $w \in (\mathbb{K} \cup \perp)^N$, it outputs a list of probabilities $p(\sigma^*)$ for $\sigma^* \in (\mathbb{K} \cup \perp)$, satisfying that $\sum_{\sigma^* \in (\mathbb{K} \cup \perp} p(\sigma^*) = 1$, at most one $\sigma \in \mathbb{K}$ has $p(\sigma) > 0$, and $p(x_i) + 0.5p(\perp) >$*

$1 - \delta(w, \mathsf{LDC}(x)) - \varepsilon$ *(resp.* $p(\mathsf{LDC}(x)_i) + 0.5p(\perp) > 1 - \delta(w, \mathsf{LDC}(x)) - \varepsilon$*) with high probability in* n*. Here, the Hamming distance between* $\perp$ *and* $\sigma \in \mathbb{K}$ *is* 0.5*. Moreover, the decoding algorithm queries any specific index with probability at most* $\frac{1.1Q}{N}$*.*

The proof can be found in the full version of the paper on arXiv.

4 Stream decodable codes of near quadratic length

In this section we prove Theorem 1, which is a construction of a stream-decodable code that achieves nearly quadratic length in polylogarithmic space. We restate it here:

▶ **Theorem 1.** *Fix any absolute constant* $\varepsilon > 0$*. Then, for some large enough* $C = C(\varepsilon)$ *and* $c = c(\varepsilon)$*, the following hold.*

- *If* $s(n) = (\log n)^t$ *for some absolute constant* $t > C$*, then there is a* $\left(\frac{1}{4} - \varepsilon, n^{2+c/t}, s(n)\right)$*-stream coding scheme.*
- *For any function* $s(n) = (\log n)^{\omega(1)}$*, there is a* $\left(\frac{1}{4} - \varepsilon, \frac{n^{2+o(1)}}{s(n)}, s(n)\right)$*-stream coding scheme. (Here, the implicit constants in the* $o(1)$ *may depend on those in the* $\omega(1)$*.)*

Both schemes succeed with probability $1 - \frac{1}{n^{\omega(1)}}$*.*

For convenience, we will scale ε by a factor of 10, so that the adversary introduces at most $(1/4 - 10\varepsilon)m$ errors into the stream (rather than $(1/4 - \varepsilon)m$). Also, we will present an algorithm whose space is $O(s(n))$ rather than just $s(n)$, since we can simply scale $s(n)$ to account for this. We will assume throughout this section that n is sufficiently large.

The encoding algorithm enc will be very simple. First, we let $Q = \min(s(n)^{0.1}, 2^{\sqrt{\log n}})$. Then, we take a code LDC as in Theorem 7 with parameters n, ε, Q, and let $y = \mathsf{LDC}(x)$ have length $N = n \cdot (\log_Q n)^{O(\log_Q n)}$. Then, we simply define enc as follows:

▶ **Definition 9.** *Define* $\mathsf{enc}(x) = \mathsf{LDC}(x)^k = y^k$*, where* y^k *denotes the string* y *repeated* k *times, and* $k = Q^2 n/s$*.*

Note that we will then have $m := |\mathsf{enc}(x)| = kN$, so the adversary will be permitted at most $(1/4 - 10\varepsilon)kN$ errors.)

We now describe how the decoding algorithm dec will work, as given formally in Algorithm 1. Let u be chosen as in the "local decoding with advice" property in Theorem 7, and let $v = (\log n)^2$. Pick indices $j_1, \ldots, j_{u+v} \in [1, N]$, where $j_1, \ldots, j_u$ are chosen according to the distribution $\mathcal{D}$ in Theorem 7 and $j_{u+1}, \ldots, j_{u+v}$ are chosen uniformly randomly.

Informally, these bits will be used to create a "checksum" for $y = \mathsf{LDC}(x)$. More explicitly, we will use the first part of the stream to determine what $y_{j_1}, \ldots, y_{j_{u+v}}$ are with high probability. Then, for the remaining (corrupted) copies of y, we will first check whether they have at most $1/4 - \varepsilon$ errors by comparing their bits at $j_{u+1}, \ldots, j_{u+v}$. Then, if they do, we will use the local decoding with advice property in Theorem 7 to recover bits of x.

More specifically, for each index j_t, we will keep track of quantities P_0^t, P_1^t, which are the total confidence that y_{j_t} is 0 or 1, respecitively. When receiving the stream, we will perform smooth local decoding (as described in Theorem 7) on each (corrupted) copy of y, to obtain confidences p_0, p_1 for each index j_t (in parallel for each t). For each t, we then increment P_0^t, P_1^t, respectively, by the obtained p_0, p_1. We show the following claim:

▷ Claim 10. With high probability, the following holds for all t and at every step of the algorithm: Let $b = y_{j_t}$. Suppose that, after reading ℓ corrupted copies of y, we have $P_b^t \le (1 - \varepsilon)k/2$. Then, the number of errors in the first ℓ copies of y is at least $\frac{1}{2}(1 - \varepsilon)(\ell - \frac{1}{2}k)N$.

The proof can be found in the full version of the paper on arXiv.

Algorithm 1 Decoding algorithm dec for Theorem 1.

Let $Q = \min(s^{0.1}, 2^{\sqrt{\log n}})$, $k = k = Q^2 n/s$.
$\varepsilon \leftarrow \varepsilon/10$.
Sample $j_1, \ldots, j_a \sim \mathcal{D}$, where $\mathcal{D}$ is as in Theorem 7.
Sample $j_{u+1}, \ldots, j_{u+v} \in [N]$ uniformly randomly.
For each $1 \le t \le u+v$, set $P_0^t, P_1^t \leftarrow 0$.
while there exists t such that $P_0^t, P_1^t \le (1-\varepsilon)k/2$ **do**
    Read next copy $y^{(i)}$, and perform smooth local decoding on $y^{(i)}$ as in Theorem 7, $u+v$ times in parallel, to get confidences p_0^t, p_1^t for each index j_t.
    For each t, set $P_0^t \leftarrow P_0^t + p_0^t$ and $P_1^t \leftarrow P_1^t + p_1^t$.
For each t, let b_t be such that $P_{b_t}^t > (1-\varepsilon)k/2$.
while there are bits remaining to output **do**
    Read the next copy $y^{(i)}$, keeping a counter c tracking how many $u < t \le u+v$ satisfy $y_t^{(i)} = b_t$. ▷ *We will prove that the copies will not run out before decoding is complete.*
    In parallel with the above, perform local list decoding with advice as in Theorem 7 using $y_{j_t} = b_t$ for $1 \le t \le u$, in parallel for the next $r = s(n)/Q^2$ bits that have not yet been output.
    if $c < (1/2 - \varepsilon)v$ **then**
        Output the decoded bits.

5 Stream decodable codes require quadratic length

We now prove our lower bound, Theorem 2 (restated below), demonstrating that the construction in Section 4 is essentially tight. That is, any error-correcting code that can be decoded with failure probability at most $1/2n^2$ by a stream permitting $s(n)$ space must have encoding length at least $\Omega\left(\frac{n^2}{s(n)}\right)$.

▶ **Theorem 2.** *Fix an absolute constant $\rho > 0$ and let the space for the decoding algorithm be $s(n) \ge \log n$. Suppose there is a $(\rho, m, s(n))$-coding scheme for streams that succeeds with probability at least $1 - \frac{1}{2n^2}$. Then, $m = \Omega\left(\frac{n^2}{s(n)}\right)$.*

Proof. Suppose otherwise; that is, suppose that there is a (ρ, m, s)-coding scheme for streams, where $m = \rho n^2/10^4 s$, ρ is a fixed constant, and $s = s(n) \ge \log n$ (and n is sufficiently large). Also, we may assume that $s < n/100$ (otherwise the statement is obvious).

We will then demonstrate how to construct an adversarial input for this coding scheme, so that dec fails with probability at least $1/2n^2$. First, note that we can assume that dec does not output anything when it receives a 0 bit (except at the end of the stream): instead, we may have dec keep track of the length of the current run of 0s (using only $O(\log n) \lesssim s$ memory), and process all the 0s when it encounters the next 1. In particular, we will have the adversary replace several parts of the input with all 0s, and thus we can assume that the algorithm does not output anything at these parts (except perhaps the last block).

For an input string $x \in \{0,1\}^n$, the encoding $\mathsf{enc}(x)$ then has length m. We split $\mathsf{enc}(x)$ up into $k = n/100s$ contiguous "blocks" which each consist of $\ell = \rho n/100$ bits; denote these blocks $B_1(x), \ldots, B_k(x)$ (we will sometimes abbreviate $B_i(x)$ by just B_i). We will consider what the algorithm dec may output during each block, assuming that the block B_i is uncorrupted. Essentially, we will show that there is a fixed set of roughly ℓ indices such that dec essentially only outputs indices from this set while it is processing block i.

To this end, we will let $a_i \in \{0,1\}^s$ denote the contents of the memory of dec right before receiving block i. Note that a_i is random and may depend on the randomness of dec, as well as on the bits that the adversary has changed in previous blocks. We will mostly restrict our attention to values of a_i that do not cause the algorithm to fail with significant probability. Specifically, we say that a_i is *good with respect to* x, or just *good* (for particular values of x and i), if the probability that dec outputs an incorrect bit during block i with starting memory a_i is at most $1/n^2$. (This probability is taken over only the randomness of the algorithm dec, since the contents of block B_i are a deterministic function of x.)

Now, suppose that the decoder dec currently has memory state a_i and is about to receive block i (whose contents are B_i). While it processes B_i, it will output various bits of x; that is, there are various pairs (j,b) for which dec will output that $x_j = b$. (We assume, as we may, that dec keeps track of which index it is on, so we can determine j from the memory contents of dec.) When it does so, we say that dec outputs the pair (j,b). Then, let $T(a_i, B_i)$ be the set of all (j,b) which dec outputs with probability at least $1/n^2$ when it receives B_i with initial memory contents a_i. (Again, this probability is only over the randomness of dec.) Note that if a_i is good (with respect to x), then $T(a_i, B_i)$ must match x (that is, $x_j = b$ for all $(j,b) \in T(a_i, B_i)$). Note that $T(a_i, B_i)$ is a *deterministic* function of a_i, B_i.

We are now ready to prove the following lemma.

▶ **Lemma 11.** *There exists $x \in \{0,1\}^n$ such that the following holds for all i: There is a set S_i of size at most 3ℓ such that for all good a_i, we have $|T(a_i, B_i(x)) \setminus S_i| < 3s$.*

Proof. Let $a_i^{(1)}, \ldots, a_i^{(r)}$ each be good a_i (with respect to a particular choice of x and i), where $r = \ell/s$. Consider the following union:

$$\mathcal{T} = \bigcup_{1 \le j \le r} T(a_i^{(j)}, B_i).$$

Essentially, if we can show, for a particular x, that this union is always small, we will then be able to show that $T(a_i, B_i)$ cannot take too large a range of values as (good) a_i varies, because the union of any r such instances will have small size. To this end, we first show the following claim.

▷ Claim 12. There exists x such that, for every i, the union $\mathcal{T}$ always has size at most 3ℓ (no matter the choice of r good $a_i^{(j)}$'s).

Proof. First observe that since each $T(a_i, B_i)$ must match x, this means that $\mathcal{T}$ must also match x (recall that this means that for every $(j,b) \in \mathcal{T}$, we have $x_j = b$). However, $\mathcal{T}$ is a deterministic function of $(B_i, a_i^{(1)}, \ldots, a_i^{(r)})$, which consists of 2ℓ bits. Therefore, there are only at most $2^{2\ell}$ possible values that $\mathcal{T}$ can take for any particular i. Thus, in total (over all i) there are at most $n \cdot 2^{2\ell} < 2^{3\ell}$ possible values for $\mathcal{T}$.

However, each possible value of $\mathcal{T}$ that has size at least 3ℓ can only match a $2^{-3\ell}$ proportion of $x \in \{0,1\}^n$. Therefore, there exists some x which does not match any possible $\mathcal{T}$ of size at least 3ℓ, thus proving the claim. ◁

Now fix x such that Claim 12 holds, and let i be arbitrary. We will now finish the proof of Lemma 11 by constructing S_i. Let $\mathcal{F}$ be the family that consists of $T(a_i, B_i)$ for all good a_i. Claim 12 means that the union of any r sets in $\mathcal{F}$ has size at most 3ℓ. We wish to find S_i which contains all but at most $3s$ elements of each $T \in \mathcal{F}$.

Now, construct S_i in steps as follows: at each step, find $T \in \mathcal{F}$ which has more than $3s$ elements which are not in S_i, and add all its elements to S_i. This process terminates when there is no such T remaining. Obviously this set satisfies $|T \setminus S_i| \leq 3s$ for all $T \in \mathcal{F}$, so it remains only to check that $|S_i| < 3\ell$. Indeed, suppose that $|S_i| \geq 3\ell$; consider the first step in which its size reached or exceeded 3ℓ. Note that at each step the size of S_i increases by more than $3s$, so in total the number of steps for $|S_i|$ to reach 3ℓ is at most $\frac{3\ell}{3s} = r$. But then S_i is the union of at most r sets in $\mathcal{F}$ and has size at least 3ℓ, contradicting Claim 12. Thus, S_i has the desired properties, completing the proof of Lemma 11. ◀

With this lemma proven, we return to the proof of Theorem 2. We will now demonstrate a strategy for the adversary such that, with probability at least $1/2n^2$, the decoding algorithm dec fails to output x. Fix the input x and sets S_i such that Lemma 11 is satisfied.

Now, the adversary picks a uniformly random index $j \in \{1, \ldots, n/2\}$. Then, for each i such that S_i contains at least $5s(n)$ indices in $[j+10(i-1)s, j+10is)$, the adversary replaces the whole block B_i with zeros (unless it is the last block). We will first show that dec must fail on this input with probability at least $1/2n^2$. Let us suppose otherwise.

As before, let a_i be the (random) memory state of dec before processing B_i. Note that with probability at least $1/2$, all the a_i are good (since in the cases where they are not, dec fails with probability at least $1/n^2$). If they are all good, then by the definition of T and a union bound (over the block number i and the indices j), with probability at least 0.99, at every block B_i, the indices output during block i are all in $T(a_i, B_i)$. In this case, observe that during block i, dec may never output the index $j + 10is$ (or any greater index). Indeed, if this were not the case, during some block dec would have to output everything in $[j + 10(i-1)s, j + 10is)$, but then we would have $|T(a_i, B_i) \setminus S_i| > 5s$, contradicting Lemma 11.

Thus, right before the last block, the algorithm cannot have output any index past $j+10ks = j+n/10$. Then, in the last block, the algorithm outputs at most $|S_i|+3s \leq 3\ell+3s$ by Lemma 11. Therefore, overall, with probability at least 0.49, the algorithm outputs at most $j + n/10 + 3\ell + 3s < n$ indices, and thus does not output all of x.

Therefore we have shown that, under this strategy for the adversary, the algorithm must fail on x with probability at least $1/2n^2$. It remains only to show that the adversary deletes (i.e., replaces with 0's) at most an ρ fraction of blocks. Indeed, it is enough to show that at most an ρ fraction of blocks are deleted in expectation, since the adversary can pick j such that the fewest blocks are deleted. Fix a block B_i. The probability that B_i gets deleted is equal to the probability that S_i has at least $5s$ indices in the interval $[j + 10(i-1)s, j + 10is)$. For any fixed $j' \in B_i$, there are at most $10s$ choices of j such that j' lands in this interval, so the probability that it does is at most $20s/n$ (since j is chosen uniformly at random from $n/2$ choices). Thus the expected number of indices in S_i in the interval is $(20s/n)|S_i| \leq 60s\ell/n$. By Markov's inequality, the probability that this is at least $5s$ is at most $12\ell/n < \rho$. Therefore, the probability that B_i is replaced with 0's is at most ρ for each block B_i. The expected number of blocks replaced by 0's is therefore at most ρk.

Putting everything together, the adversary has a strategy which deletes at most an ρ fraction of blocks (and thus at most an ρ fraction of the bits) which causes dec to fail with probability at least $1/2n^2$. This completes the proof of Theorem 2. ◀

6 Stream decodable codes for linear functions of near linear length

Our final result is a noise-resilient encoding of essentially linear length that admits efficient stream decoding of arbitrary linear functions. The family of linear functions is defined as the functions $f : \{0,1\}^n \to \{0,1\}$ for which there exists $y \in \{0,1\}^n$ such that $f(x) = x \cdot y \mod 2$.

▶ **Theorem 3.** *Fix any absolute constant $\varepsilon > 0$. Then, for some large enough $C = C(\varepsilon)$ and $c = c(\varepsilon)$, the following hold.*

- *If $s(n) = (\log n)^t$ for some absolute constant $t > C$, then there is a $\left(\frac{1}{4} - \varepsilon, n^{1+c/\sqrt{t}}, s(n)\right)$-stream coding scheme for the family of linear functions.*
- *If $s(n) = \Omega(n^\delta)$ for some absolute constant $\delta > 0$, then there is a $\left(\frac{1}{4} - \varepsilon, O(n), s(n)\right)$-coding scheme for the family of linear functions.*

Both schemes succeed with probability $1 - \frac{1}{n^{\omega(1)}}$.

Parameters and notation

Throughout this section, fix ε (we will hide dependence on ε in big O notation), δ if it exists, and the space function s. Let n represent the length of Alice's message. We assume that $s(n) > (\log n)^{1000}$. Set the following parameters:

$$r = s(n)^{0.2} \quad \text{and} \quad d = \frac{\log n}{\log r} \quad \text{and} \quad \varepsilon' = \frac{\varepsilon}{10d} .$$

Let $\mathbb{K}$ be $\mathbb{F}_{2^k}$ where $\varepsilon'^{-10} < 2^k \leq 2\varepsilon'^{-10}$ so that the condition of Theorem 8 is satisfied for $\mathbb{K}$ and ε'.

Let $\mathsf{LDC} : \mathbb{K}^r \to \mathbb{K}^R$ with be a linear locally decodable code satisfying the guarantees of Theorem 8 for ε'. The locality is $Q > \left(\varepsilon'^{-1} \log r\right)^{100}$. This is satisfied whenever $Q > (d \log r)^{1000}$ because

$$\left(\varepsilon'^{-1} \log r\right)^{100} \leq \left(d\varepsilon^{-1} \log r\right)^{100} \leq \left(d(\log r)^2\right)^{100} \leq (d \log r)^{1000}$$

since $\log r$ is sufficiently large compared to ε^{-1}. We will set Q subject to this constraint later. Also note that $Q > \varepsilon^{-1} \log n = \varepsilon^{-1} d \log r$ which is a fact we will use later and $Q < r$ must be satisfied.

This gives us a value of $R \geq (\varepsilon'^{-1} \log_Q r)^{100 \log_Q r}$, satisfied if $R \geq (d \log_Q r)^{150 \log_Q r}$. We will actually set R later, subject to this constraint. We can make R larger as needed by a variety of methods, such as padding the input or duplicating each bit of the code.

We assume for simplicity that r and d are integers. It will be useful to index Alice's (the sender's) input $x \in \{0,1\}^n$ by a tuple in $[r]^d$ rather than an integer in $[n]$. Whenever we say an event occurs with high probability, we mean with high probability in n unless specified otherwise. We refer the reader to Section 2 to review notation used in this section.

6.1 Statement of encoding scheme

The encoding $\mathsf{enc}(x)$ that Alice (the encoder) sends in the stream is a tensor code. To this end, we begin by defining a tensor power of a linear code C. We remark that tensor products of distinct linear codes can also be defined, but we will only need to take a tensor power of one code.

▶ **Definition 13** ($\mathsf{C}^{\otimes k}$). *Let $\mathsf{C} : \mathbb{K}^m \to \mathbb{K}^M$ be a linear error correcting code on strings of length $N \in \mathbb{N}$ on some alphabet $\mathbb{K}$. Since the code is linear, C is an $M \times m$ matrix. Then the k-th tensor power of this encoding matrix $\mathsf{C}^{\otimes k} : \{0,1\}^{[m]^k} \to \{0,1\}^{[M]^k}$ is the encoding function $\mathsf{C}^{\otimes k}$. We note that for any code C, it holds that $\mathsf{C}^{\otimes 0} : \mathbb{K} \to \mathbb{K}$ is the identity function.*

Next, we will state the encoding $\mathsf{enc}(x)$ that Alice (the encoder) sends in the stream.

▶ **Definition 14** (C_{inner}). *Let $\mathsf{C}_{inner} : \mathbb{K} \to \{0,1\}^{O(1)}$ be a distance $(1-\varepsilon/4)$ linear code guaranteed by Theorem 5. It's length is N_{inner}.*

▶ **Definition 15** ($\mathsf{enc}(x)$). *Alice's encoding is $\mathsf{enc}(x)$ is defined as follows. Viewing x as an element of $\mathbb{K}^n$ (which we may since $\mathbb{K}$ is of the form $\mathbb{F}_{2^k}$, she computes $\mathsf{LDC}^{\otimes d}(x)$ (where the message and codeword bits are both taken in lexicographic order) and concatenates this with C_{inner}.*

We note that Alice's encoding is length R^d, and her encoding takes time at most $\mathrm{poly}(n)$. We'll later choose our parameters to satisfy the conditions of Theorem 3.

6.2 Statement of decoding scheme

Let Bob's private vector be $\ell = \langle \ell_{(1,\dots,1)}, \dots, \ell_{(r,\dots,r)} \rangle$, (here the ordering is lexicographic). The function Bob is trying to compute is $\ell \cdot x = \ell_{(1,\dots,1)} x_{(1,\dots,1)} + \dots + \ell_{(r,\dots,r)} x_{(r,\dots,r)}$. For $a \le d$, the vector $\ell_{(i_1 \dots i_{d-a})}$ is defined to be an r^a dimensional vector that denotes ℓ restricted to indices where the first $d-a$ entries of the tuple are $(i_1 \dots i_{d-a})$. Throughout, we will view ℓ and x as elements of $\mathbb{K}$ rather than $\mathbb{F}_2$ and note that it suffices to compute $\ell \cdot x$ in $\mathbb{K}$. The same notation applies for x or any other string canonically indexed by tuples.

Before we state our main algorithm, we efficiently construct lists $\mathsf{qlist}_1, \dots, \mathsf{qlist}_r$ satisfying certain properties that Bob can find in $s(n)$ space and $\mathrm{poly}(n)$ time. These lists will be the indices of LDC that we query for each i, and it will be important that they overlap as little as possible.

▶ **Lemma 16.** *Given a locally decodable code $\mathsf{LDC} : \mathbb{K}^m \to \mathbb{K}^M$ satisfying $m > \log n$ with q queries satisfying the requirements of Theorem 8 for ε', there is a (randomized) algorithm (permitted $\frac{1}{m^{\omega(1)}}$ probability of failure) that generates lists $\mathsf{qlist}_1, \dots, \mathsf{qlist}_m$ in time $\mathrm{poly}(mq)$ and space $O(mq^2)$ such that the following holds. The smooth local decoding algorithm for each index i queries only the indices qlist_i, and for each $z \in \{0,1\}^M$, with high probability in m, satisfies the smoothness guarantees in Theorem 8 for all i. Moreover, no $I \in [M]$ appears in more than $\left\lceil \frac{3mq^2}{M} \right\rceil$ lists.*

The proof can be found in the full version of the paper on arXiv.

We are now ready to state the main algorithm.

Algorithm 2 Bob's decoding algorithm linear_dec.

input: $n, d \in \mathbb{N}$, $\mathsf{LDC} : \{0,1\}^r \to \{0,1\}^R$, and stream $z \in \{0,1\}^{R^d}$.

function recurse_linear(integer $a \le d$, stream $z \in \{0,1\}^{N_{inner}R^a}$, tuple $T \in [r]^{d-a}$):
  if $a = 0$ **then**
    Set σ to the decoding of z via C_{inner} with probability $1 - 2\delta(z, \mathsf{C}_{inner}(\sigma))$ and otherwise to $\perp$. This is the only step where the stream is read, and it happens in unison for all instances reading this stream; the randomness in deciding σ is the same for all of them.
    return $\ell_T \cdot \sigma$ ($\perp$ if $\sigma = \perp$).
  Pick lists $\mathsf{qlist}_1, \ldots, \mathsf{qlist}_r$ as in Lemma 16.
  Define maps $\mathcal{M}_i : \mathsf{qlist}_i \to \mathbb{K}$ for $i \in [r]$ as follows:
  for $I \in [R]$ **do**
    Let z_I denote the next R^{a-1} elements of the stream z.
    for i such that $I \in \mathsf{qlist}_i$ **do**
      Set $\mathcal{M}_i[I] \leftarrow \mathsf{recurse\_linear}(a-1, z_I, T|i)$.
  for $i \in [r]$ **do**
    Compute $p_i(\sigma)$ for all $\sigma \in \mathbb{K}$ using local decoding on index i with queries $\mathcal{M}_i$.
    Set the guess $\sigma_i = \arg\max_\sigma p_i(\sigma) \in \mathbb{K}$ and likelihood $p_i = \max_\sigma p_i(\sigma)$.
  Let $p = \min_i p_i$.
  return $\sigma_1 + \ldots + \sigma_r$ with probability p and otherwise $\perp$.

Compute $\mathsf{recurse\_linear}(d, z, ())$ in parallel $(\log n)^2$ times. When the output is $\perp$, assign it 0 or 1 randomly and otherwise project to $\mathbb{F}_2$. Let majority output be σ.

output: σ.

The proof of Theorem 3 can be found in the full version of the paper on arXiv.

References

1 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. 6

2 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. 7

3 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. 6

4 Jiecao Chen and Qin Zhang. Bias-aware sketches. *arXiv preprint arXiv:1610.07718*, 2016. arXiv:1610.07718. 7

5 Zitan Chen, Sidharth Jaggi, and Michael Langberg. A characterization of the capacity of online (causal) binary channels. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing*, pages 287–296, 2015. doi:10.1145/2746539.2746591. 6

6 Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM Journal on Computing*, 40(4):1154–1178, 2011. doi:10.1137/100804322. 6

7 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. 6

8 Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard J. Schulman. Optimal coding for streaming authentication and interactive communication. *IEEE Transactions on Information Theory*, 61(1):133–145, 2015. doi:10.1109/TIT.2014.2367094. 6

9 Sumegha Garg, Pravesh K Kothari, Pengda Liu, and Ran Raz. Memory-sample lower bounds for learning parity with noise. *arXiv preprint arXiv:2107.02320*, 2021. arXiv:2107.02320. 7

10 Ran Gelles. Coding for interactive communication: A survey. *Found. Trends Theor. Comput. Sci.*, 13(1-2):1–157, 2017. doi:10.1561/0400000079. 3

11 Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N Rothblum. Verifying and decoding in constant depth. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 440–449, 2007. doi:10.1145/1250790.1250855. 8

12 Meghal Gupta and Rachel Yun Zhang. A noise resilient transformation for streaming algorithms. *arXiv preprint arXiv:2307.07087*, 2023. doi:10.48550/arXiv.2307.07087. 3, 5, 8

13 Venkatesan Guruswami and Valentine Kabanets. Hardness amplification via space-efficient direct products. *Computational Complexity*, 17(4):475–500, 2008. doi:10.1007/S00037-008-0253-1. 5, 6

14 Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at http://cse. buffalo. edu/faculty/atri/courses/coding-theory/book*, 2019. 7

15 Venkatesan Guruswami and Adam Smith. Optimal rate code constructions for computationally simple channels. *Journal of the ACM (JACM)*, 63(4):1–37, 2016. doi:10.1145/2936015. 6

16 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. 7

17 R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950. doi:10.1002/j.1538-7305.1950.tb00463.x. 2

18 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. 6

19 Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. *J. ACM*, 61(5), September 2014. doi:10.1145/2629416. 6

20 Swastik Kopparty, Ronen Shaltiel, and Jad Silbak. Quasilinear time list-decodable codes for space bounded channels. In *60th Annual Symposium on Foundations of Computer Science (FOCS)*, 2019. 6

21 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. 7

22 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. 6, 7

23 Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978. doi:10.1145/359619.359627. 6

24 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(3):6–12, 1954. doi:10.1109/IREPGELC.1954.6499441. 6

25 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. 6

26 Leonard J. Schulman. Coding for interactive communication. *IEEE Trans. Inf. Theory*, 42(6):1745–1756, 1996. doi:10.1109/18.556671. 2, 3

27 Ronen Shaltiel and Jad Silbak. Explicit list-decodable codes with optimal rate for computationally bounded channels. *computational complexity*, 30:1–70, 2021. 6

28 Ronen Shaltiel and Jad Silbak. Explicit uniquely decodable codes for space bounded channels that achieve list-decoding capacity. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1516–1526, 2021. doi:10.1145/3406325.3451048. 6

29 C. 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. 2

30 Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Inf. Theory*, 42(6):1723–1731, 1996. doi:10.1109/18.556668. 5, 6

31 Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 537–546, 1999. 8

32 Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012. doi:10.1561/0400000030. 5

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